

# CHAPTER 1

## 'C' Basics

### History of 'C'

Before 1960 there were so many languages in existence to solve the different types of problems. COBOL was there for business related applications and FORTRAN was used for mathematical applications. No common language was there to solve all the problems. To solve this problem, a committee came into existence which introduced a new language ALGOL – 60 but it seemed very abstract and general. To reduce this abstractness and generality they came up with a new language named CPL (Common Programming Language) but it was too hard to learn because it was too specific and there were so many drawbacks in this language. To remove these problems BCPL (Basic common programming language) was made by the Martin Richards of Cambridge University. But it was less powerful and too specific. After this “B” language was developed by Ken Thompson at AT & T Lab. It was interpreter based and slow.

Dennis Ritchie of AT & T Bell Labs developed 'C' with some modifications in B language and BCPL.

**('C' Language Was Developed by Dennis Ritchie of AT & T Bell Laboratories in 1972.)**

## 'C' Scope

### Why 'C' is so powerful?

There are many reasons, and is mainly due to the fact that it is a one man language which makes 'C' a very powerful language for programming

- a. It has facilities of a high level language as well as of low level language.
- b. Rich in many functions.
- c. 'C' is machine independent and hence it's a portable language.
- d. It can be used for commercial as well for scientific and Graphical works
- e. It is also used in system programming.

f.

### Different Versions of 'C'

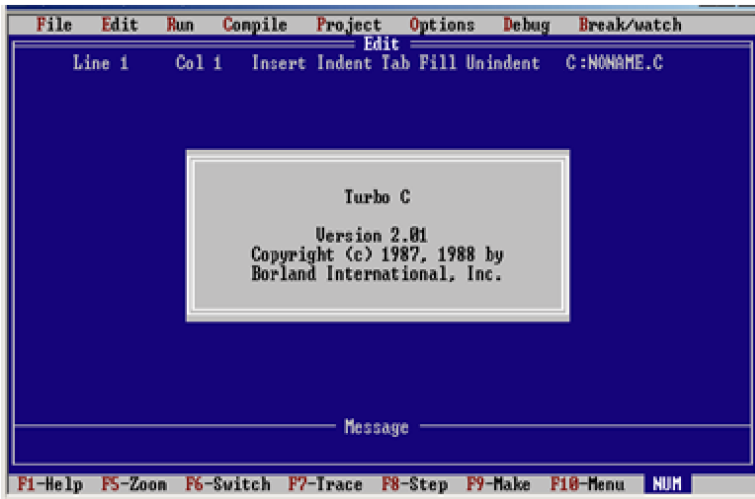
1. Microsoft 'C'
2. Ansi 'C'
3. Turbo 'C'
4. Quick 'C'
5. Borland 'C'

Among all the above versions only Ansi 'C' is Unix Based 'C' and all others are DOS or windows based.

### Introduction to Turbo 'C' IDE

Turbo C is an integrated development environment & a compiler, developed by Borland company for C programming.

## 'C' Scope



Turbo C version 1.0 was out in the market in 1987, which was followed by versions 1.5 and 2.0, in 1989 it was the most popular compiler for C, developed in MS-DOS framework.

It is first IDE available for C platform. It was replaced by Turbo C++ IDE in 1990.

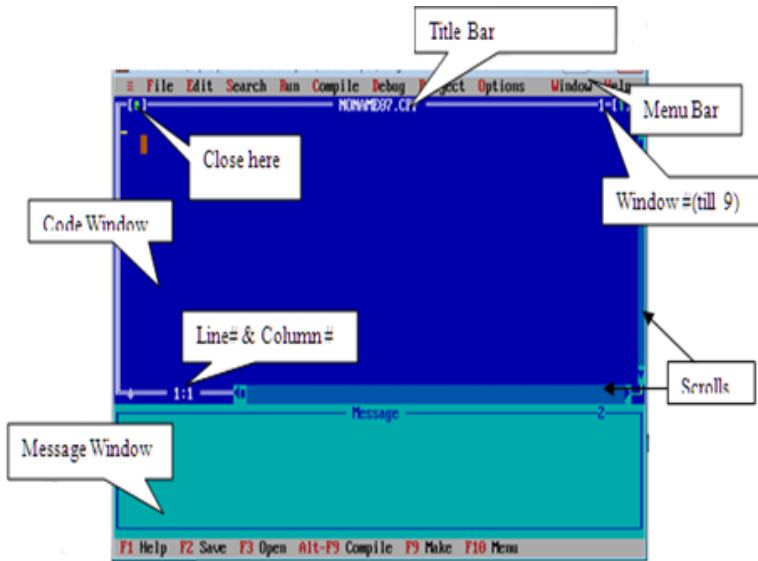
### **Introduction to TURBO C++ IDE**

Turbo C++ is a Borland C++ compiler with an integrated IDE. Turbo C++ was a successor of Turbo C, expanding the compiler functionality further.

Turbo C++ 3.0 was released in 1992, and came in amidst expectations of the coming release of Microsoft Windows 3.1. Turbo C++ v3.0 first came as an MS-DOS compiler, supporting C++ templates, generation of DOS & protected

## 'C' Scope

mode executables.



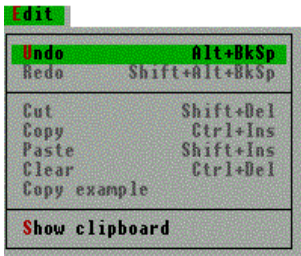
This menu help you to create a new file or open an existing one and it also provide option to save or print it.

While working in this compiler you can also shift to DOS Prompt using DOS Shell option

Here you will also find a Quit option to come out of the compiler.

## 'C' Scope

---



This menu helps in editing the program, doing cut, copy, paste. Also you can revert & redo the changes done in our program.



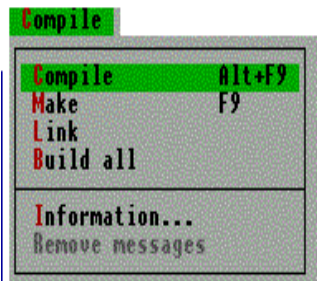
This menu helps in maneuvering in the whole program.



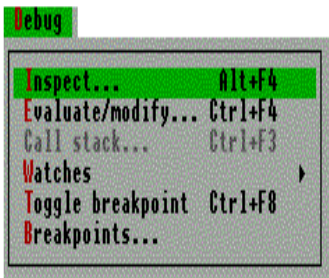
The run menu contains all the options used for & during execution of the program. Its various options like Trace into & Step over helps in tracing the transfer of control in the program.

## 'C' Scope

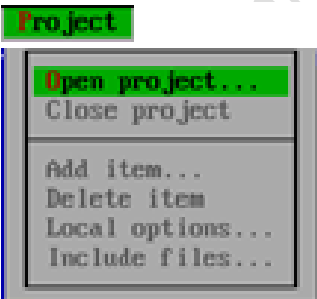
---



The Compile menu contains all the options related with compiling, linking & loading of the program.



This menu facilitates the user to use various debugging tools during removing errors from the program. You can add watches, breakpoints to your code and also can evaluate the code.



You can also make project in C Language like in other Language's environment, it also supports making of .exe file that is an executable file of the whole project.

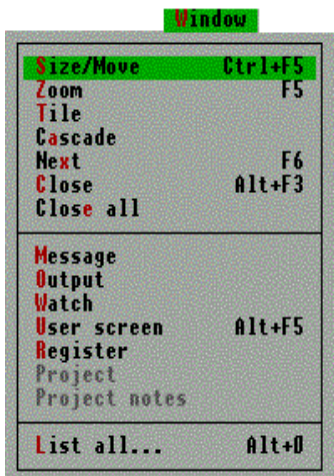
---

## 'C' Scope

---



In option menu various options like changing compiler settings, making changes in Environment, applying options of linking to Library files.



This menu contains various options for changing window & screen options.

## 'C' Scope

---



Help menu contains options to access Turbo C++ IDE help.

---

### Some general commands for the use in Turbo 'C++'

#### IDE

F1	Help
F2	To Save
F3	Open an Existing File
Alt F3	To Close opened file
F5	Maximize
F6	Toggle between Windows
Alt F5	To go to the output Screen
^y	To delete entire line
^t	To delete a entire word
^Qy	To delete rest of line
Alt+X	To come out from 'C' Ide
^F9	To compile and Run
F9	To compile
^qr	To go to the beginning of file
^qc	To go to the end of the file
^qf	To find string
^qa	To find and replace
Alt Enter	To restore the window
Alt Backspace	To Undo
^ins	To Copy
Shift Insert	To Paste



## 'C' Scope

---

Shift del	To Cut
^del	To Clear Screen (all)
^F1	For Help about Topics

### 'C' Character Set

Character set in 'C' denotes all the characters that have some valid meaning in 'C', or contains any specific meaning in 'C' language.

All the valid characters in 'C' are as follows.

Alphabets	:	A-Z , a-z.
Numerals	:	0-9
Operators	:	- + * / % (MOD)
Special Symbols	:	{ } ( ) , . ; ' " ~ ? : < > = & ^   { [ ]

### 'C' Tokens

The smallest individual words with or without symbols that contains some particular meaning in C language are called C Tokens.

### Variables / Identifiers

Variable are those meaningful names given to computer's memory location that are used to store data. When using variable you refer to memory location that contains data value.

## 'C' Scope

### Rules for Variable Naming

There are certain rules that have to be followed for naming a variable.

1. A variable name can be up to 32 characters long.
2. Only first 32 characters are significant in variable name. Like in valid variable names given below contain ( names the\_average\_height\_of\_any\_person\_in\_india , the\_average\_height\_of\_any\_person\_in\_usa ) when taken in different program it would work, and iff taken in the same program it would give error because first 32 characters are same.
3. We can use main as a name of any variable.
4. Variable name is case sensitive, So variable name, "age" is different from "AGE" .
5. It can be a combination of all alphanumeric characters but it must start with any alphabet or underscore.
6. Special symbols are not allowed while constructing a variable name.
7. Spaces are not allowed in variable name.
8. Keywords are not allowed in a variable name. But keywords of only C++ can be variables in 'C'.
9. Variable name can't be pre-defined variables of included Header File.

### These are the valid variables names

Check\_num  
\_digit  
pie1

## 'C' Scope

`_int`  
`val2`  
`sno`  
`x10`  
`first_digit`  
`Value_of_factorial`  
`i_am_resident_of_india_in_the_universe`  
`i_am_resident_of_india_in_the_uuniverse_in_rajasthan`

### Invalid variable names

`2sum` (variable name cannot start with a number)  
`d.val` (dot . is not allowed in a variable name)  
`check sum` (variable name cannot contain spaces)  
`long` (variable name cannot use reserved words)  
`check%d!54` (variable name cannot include special symbols)  
`EOF` (variable name can't be pre-defined variables of included header file & this is defined in `stdio.h` header file)

### Keywords

The words that have predefined meaning in 'C' are known as key words also called as reserved words, and cannot be used for other purpose in programming.

The listing of keywords or reserved words is given below

<code>auto</code>	<code>double</code>	<code>int</code>	<code>struct</code>
<code>break</code>	<code>else</code>	<code>long</code>	<code>switch</code>
<code>case</code>	<code>enum</code>	<code>register</code>	<code>typedef</code>
<code>char</code>	<code>extern</code>	<code>return</code>	<code>union</code>

## 'C' Scope

const	float	short	unsigned
continue	for	signed	volatile
default	goto	sizeof	void
do	if	static	while

Table 1) Shows Reserved Words

### Escape Sequences

In 'C' language the meaning of escape sequence is to escape the normal meaning of a character. They are used for non printable characters in a program.

\n New line	\a Alert	\v Vertical tab
\t Tab	\f Form	\' Single quote
\b Back space	\r Carriage return	\” Double quote
\? Question mark	\\ Back slash	\0 Null

### Declaring variables in 'C'

When we use any variable in any program we have to declare them first and then only we can use it in our program in 'C'. Syntax to declare any variable is given as follows

**datatype variablename1 , ..... , variablename(n)**

### Data Types

Data type is used to declare that which type of data we are going to use in our program. And according to the data type the memory used by the variable is decided.

## 'C' Scope

A program usually contains different types of data types (integer, float, character etc.) to store the values being used in the program along with some library function and user defined function (UDF) to process that stored data. 'C' language is rich in data types and library functions. A 'C' programmer has to employ proper data type as per requirement.

'C' provides four fundamental data types.

1. int
2. char
3. float
4. double

The ranges for these data types are as under

<u>Type</u>	<u>Size (bits)</u>	<u>Range</u>
char or signed char	8	-128 to 127
unsigned char	8	0 to 255
int or short signed int	16	-32768 to 32767
short unsigned int	16	0 to 65535
long signed int	36	-2147483648 to +2147483647
long unsigned int	36	0 to 4294967295
float	32	3.4E - 38 to 3.4E +38
double	64	-1.7e308 to +1.7e308
long double	80	-1.7e4932 to +1.7e4932

## **Integer Datatype**

## 'C' Scope

An integer consists of a sequence of digits having optional signs (+) or (-). It does not contain the comma (,) or decimal (.). It can be declared & initialized as follows.

```
int x=5;  
int y=-8;
```

(The maximum value that can be stored in an integer variable is 32767. If you try to exceed this by 1 it will take the value -32768)

## **Character Datatype**

It consists of ASCII character enclosed in a pair of apostrophes. It occupies 1 byte of memory and it can be signed or unsigned. The range of signed character is -128 to 127 and for unsigned it is 0 to 255. Within this range a character data behaves like integer data and vice versa. By this I mean to say that we can get an integer ASCII equivalent of any character in this range.

Here are some characters data types.

```
char ch='a';  
char x='5';  
char y='/';
```

## **Float Datatype**

It consists of a sequence of digits and a decimal. It is necessary to put a decimal in floating point or real number. It occupies 4 bytes of memory. Here are some floating point examples.

## 'C' Scope

```
float salr=4500.50;  
float temp=45.63;
```

We can change the range of data using some modifiers these modifiers are

1. unsigned
2. long
3. short
4. signed

### **unsigned**

The unsigned tells the compiler not to use the most significant bit. Using this keyword we can increase the range to represent a large value.

### **long**

Using this we can use 32 bits to represent the integer value.

### **short**

If long data type modifier doubles the size, short on the other hand reduces the size of the data type to half. Please refer to the example of age variable to explain the concept of data type modifiers.

### **signed**

By default all data types are declared as signed. Signed modifier means that the data type is capable of storing negative or positive values.

## 'C' Scope

### Structure of sample 'C' Program

Header File Inclusion
Global constants & types
Program heading
Main Function & its type
Local Declarations
Executable Statements
Return Statements
End of Main Function

### Operators

An operator tells the compiler to perform some specific task using operands. Just like mathematical operators (+, -, /, -\*) 'C' offers some more operators for you. Now let's see what are these?

1. Arithmetic operators
2. Unary operators
3. Assignment operators
4. Conditional or Ternary operators
5. Relational operators
6. Logical operators
7. Bit wise operators
8. Special operators

#### **1. Arithmetic operators**

Several mathematical operators in 'C' just like normal mathematical operators do.



## 'C' Scope

There are 5 arithmetic operators in 'C' these are

- i.) +            Addition
- ii) -            Subtraction
- iii) \*           Multiplication
- iv) /            Division
- v) %            Modulus

For arithmetic operators operands can be integer quantities, real number, or character. For Modulus (%) operator both the operands must be numerals and the second operand must be nonzero. Similarly, for the division operator (/) the second operand must be non-zero.

(Note: There is no operator in 'C' that can be used for exponentiation. A function pow() is available for exponentiation).

Now I think there is no need to explain these operators because you are using these operators from your school life.

### **2. Unary Operators**

These operators act on only one operand.

- a. Unary Plus(+)
- b. Unary Minus(-)
- c. Increment (pre and post) ++
- d. Decrement (pre and post) --

Most commonly used operator is the unary minus, where a numerical constant, variable or expression is preceded by a minus sign. It is same as the arithmetic subtraction operator

## 'C' Scope

but arithmetic (-) operator requires two operands and unary (-) operator requires only one operand for operations.

There are two other commonly used unary operators and these are increment (++) and decrement operator (--). These operators are used to increase or decrease the value of variable by 1.

We can understand the difference between the pre and post increment or decrement operators using following program segments

```
x=10;
x++;
printf ("%d",x);
```

the output would be 11

```
x=10;
printf ("%d",x++);
```

the output would be 10

In first program segment the value of x would be 11 and in second program segment the output is 10. Because for the first example in first line the value of x is ten in next line it will be increased by one the value of x will be 11 which will be printed in third line.

For second example in first line the value of x is 10, in second line first the operation will be performed i.e the value of x will be printed which is ten and then it will be increased by one.

Like this we can also understand the differences between the pre and post operators by the following example:-

Example of pre operator:-

## 'C' Scope

```
#include<stdio.h>
main()
{
    int x,y;
    clrscr();
    x=5;
    y=++x; // Pre operator
    printf("%d",y);
    getch();
}
```

The answer will be 6

*So, pre increment or decrement operators first increase or decrease the values and then perform other operations, whereas post increment or post decrement first perform the other operation and in next the values increased or decreased by one.*

### 3. Assignment Operators

There are two types of assignment operators

- i. Normal Assignment operators
- ii. Compound Assignment operators

- i) **Normal assignment operator:** It is most commonly used operator which uses (=) sign which works exactly the same way the equal symbol does in mathematics.

E.g.

x=10;

## 'C' Scope

The value 10 will be assigned to the x;

```
a=b=c=10;
```

Here assignments are performed from the right to the left, first 10 will be from c to b to a;

ii) **Compound Assignment operators** : In 'C' language there are several compound assignment operators.

To understand these compounds assignment operators first look at the following program segment

```
x=5;
```

```
x=x+20;
```

Now I can make this code shorter in a way like

```
x=5;
```

```
x+=20;
```

Which is just shorter than the first case because we are using a compound assignment operators (+=) to make it shorter. That is why they are also called as *Shorthand Assignment Operators*.

This type of combination can be used with other operators some of these are

+=	Addition Assignment
-=	Subtraction Assignment
*=	multiplication Assignment
/=	Division Assignment
%=	Mod Assignment
<<=	Left Shift Assignment

## 'C' Scope

>>=	Right Shift Assignment
&=	Bit wise and Assignment
=	Bit wise OR Assignment
^=	Bitwise XOR Assignment

### **4. Ternary Operators (Conditional Operators) ? :**

If in any program we have to give a condition and based on that condition our program should behave accordingly that is if the condition proves out to be right then certain set of statements should be executed otherwise another set of statements should be executed. For this problem 'C' has a solution named Conditional Operators.

It contains three expressions. The conditional operator tests the first expression which is a condition in fact, if it is true then the resulting value is that of the second expressions. Otherwise, the resulting value is that of the third expressions. To understand the conditional operators look at the following program segment.

```
a=5;  
b=10;  
a > b ? printf ("New York") : printf ("Washington") ;
```

in the given segment first expression is false hence the result will be Washington.

```
a=50;  
b=20;  
c=a>b? 10: 40;
```

in the given segment the first expression is true so the value assigned to c will be 10.

## 'C' Scope

Now we can make the general form of conditional operator as

**Conditional expression? expression 2 : expression 3;**

There can be nested conditional operators to solve complex decisions.

```
per >= 60 ? printf ("First Division ") : per >= 45 ? printf ("Second Division ") : per >= 33 ? printf ("Third ") : printf ("Failed ");
```

The above example shows nesting of three conditional operators ie if any student scores percentage more than 60, first statement will be followed, otherwise goes to next condition now tests for percentage above 45 & at last above 33 & above, if all the conditional expressions results false then the snippet prints “Failed”

### **5. Relational operators**

These Operators compare expressions and returns the true or false values decided by the relative values of the expressions and the operators used, and these operators are used to check the conditions.

- |      |    |                       |
|------|----|-----------------------|
| i.   | <  | Less than             |
| ii.  | >  | Greater than          |
| iii. | <= | Less than or equal to |
| iv.  | >= | Greater than equal to |
| v.   | == | Equal to              |
| vi.  | != | Not equal to          |

## 'C' Scope

(Note: - Unlike other languages use double (=) sign for comparisons)

You should know that there lies a difference between equivalence sign(=) & relational equal to (==) operator, where the former is used for assigning values to variables and the later is used for condition matching. Let's see an example for this

```
#include<stdio.h>

void main()
{
    int a;
    clrscr();
    (a=5) ? printf("value of a is five") : printf("a is not
five");
    getch();
}
#####OUTPUT#####
value of a is five
```

In the above example the output comes out to be five because the first expression assigns value of 5 to a, and because this instruction is carried out flawlessly by the 'C' compiler so it (compiler) returns true & because of true returning the first expression is shown as result.

Instead of 5 if any other number would be there, the result would have been the same. If we really want to check the equality, we can offer the set of statements

```
#include<stdio.h>
```

## 'C' Scope

```
void main()
{
    int a=5;
    clrscr();
    (a==5) ? printf("value of a is five") : printf("a is not
five");
    getch();
}
#####OUTPUT#####
value of a is five
```

Here the value 5 is first assigned to a, and then value of 5 and value of a are compared resulting in true or false block statement as a result, unlike the above one where always the true block statement will be executed.

### **6. Logical Operators**

If you want to test more than one condition and to go for a decision, there are some logical operators offered by 'C' for you.

These three logical operators are.

- i.    &&   logical (AND)
- ii.   ||   logical (OR)
- iii.  !    logical (NOT)

(Note : Do not confuse with & and && single ampersand or single pipe signs are not used for logical decisions they have the special meaning which we will discuss further).

You can better understand these operations with the following truth table.



## 'C' Scope

expression 1	expression 2	exp1 && exp2	exp1    exp2
true	true	true	true
true	false	false	true
false	false	false	false
false	true	false	true

```
char x;  
printf("Type y or Y for yes");  
x=getchar();  
(x=='y' || x=='Y')? printf("Okay") : printf("Not Okay");
```

The above code first takes a character input from user & checks it for yes, the character entered should be either 'y' or 'Y', if the cumulative condition given proves the snippet prints " Okay " otherwise prints " Not Okay ".

### 7. Bitwise Operators

Bitwise operators in 'C' Language are used to operate on bit of any number used in the program. It is a fast, primitive action directly supported by the processor, and is used to manipulate values for comparisons and calculation on the bit level.

There are six types of Bitwise Operators

- i) Bitwise OR |
- ii) Bitwise AND &
- iii) Bitwise XOR ^
- iv) Bitwise NOT ~
- v) Bitwise Shift Left <<
- vi) Bitwise Shift Right >>

- i) **Bitwise OR (|)**

## 'C' Scope

The bitwise OR operator is used to operate logical OR operation on bit level. The OR operation truth table is explained in previous section within Logical Operators. Now, we would see how it works

Let us assume,

$$A = 6 \quad /* 0000 0110 */$$

$$B = 8 \quad /* 0000 1000 */$$

Then its Bitwise OR operation would result in

$$A | B \quad /* 0000 1110 */ \text{ i.e } 14$$

### ii) Bitwise AND (&)

The bitwise AND operator is used to operate logical AND operation on bit level. The AND operation truth table is explained in previous section within Logical Operators. Now, we would see how it works

Let us assume,

$$A = 6 \quad /* 0000 0110 */$$

$$B = 8 \quad /* 0000 1000 */$$

Then its Bitwise OR operation would result in

$$A \& B \quad /* 0000 0000 */ \text{ i.e } 0$$

### iii) Bitwise XOR (^)

The bitwise XOR operator is used to operate logical XOR operation on bit level. The XOR operation truth table is as follows.

<b>X</b>	<b>Y</b>	<b>X^Y</b>
0	0	0
0	1	1
1	0	1

## 'C' Scope

1

1

0

Let us assume,

A = 6            /\* 0000 0110 \*/

B = 8            /\* 0000 1000 \*/

Then its Bitwise OR operation would result in

A ^ B            /\* 0000 1110 \*/ i.e 14

### iv) **Bitwise NOT (~)**

The bitwise NOT operator is used to operate logical NOT operation on bit level. The NOT operation truth table is explained in previous section within Logical Operators. Now, we would see how it works.

Let us assume,

A = 6            /\* 0000 0110 \*/

Then its Bitwise OR operation would result in

~A               /\* 1111 1001 \*/

### v) **Bitwise Shift Left <<**

Bitwise Shift left operator is used to operate on bit level. This operator shifts the bits of a number one position at a time towards left.

The operation  $x \ll n$  shifts the value of  $x$  left by  $n$  bits.

Let's look at an example. Suppose  $x$  is a char and contains the following 8 bits.

If A = 11000111

Now  $A \ll 3$  would be

## 'C' Scope

⇒ 00011100

### vi) Bitwise Shift Right >>

Bitwise Shift right operator is used to operate on bit level. This operator shifts the bits of a number one position at a time towards right.

The operation  $x \gg n$  shifts the value of  $x$  right by  $n$  bits.

Let's look at an example. Suppose  $x$  is a char and contains the following 8 bits.

If  $A = 00111100$

Now  $A \gg 2$  would be

⇒ 00001111

## 8. Special Operators

The special operators in 'C' does the works are related to memory processing.

There are two types of special operators that are used in 'C' Language.

- i) sizeof Operator
- ii) Address Operator

### i) sizeof operator

The sizeof operator gives the amount of storage, in bytes, required to store an object of the type of the

## 'C' Scope

operand. This operator allows you to avoid specifying machine-dependent data sizes in your programs.

```
sizeof unary-expression  
sizeof ( type-name )
```

### ii) **Address operator**

The address of operator is used to find the memory pointer location used by any variable. The symbol used for this operator is '&'. But you will think how come the Bitwise operator(&) can be used for this purpose. But it is unary operator and can be used with only one operand. Like

```
&variable_name;
```

## **Type casting in 'C'**

If at any time in our program we want to do certain calculations using different datatype variables then 'C' language helps us in casting of one variable of some datatype into another higher datatype, whose variable is used in the expression.

Let us see an example...

```
void main()  
{  
    int rate=5,time=2;  
    float principal=5000.00,si;  
    clrscr();  
    si = (principal * rate * time)/100.00;
```

## 'C' Scope

```
    printf("Simple interest is %f",si);
    getch();
}
```

##### OUTPUT #####

Simple interest is 500.000000

In the above example rate & time variable of integer type implicitly change to float datatype to participate in the expression. This is called *implicit typecasting* which is done implicitly by the 'C' compiler.

Typecasting is simply a mechanism by which one can change the data type of a variable, no matter how it was originally defined. When a variable is type casted into a different type, the compiler basically treats the variable as of the new data type.

When the compiler does not perform the typecasting implicitly but we want that the variable should behave according to the different data type required/mentioned by the user, and then we have to perform *explicit typecasting*.

Syntax for performing that is

***(datatype) variable\_name***

The previous example if done with explicit typecasting then

```
void main()
{
```

## 'C' Scope

```
int rate=5,time=2;
float principal=5000.00,si;
clrscr();
si = (principal * (float)rate * (float)time)/100.00;
printf("Simple interest is %f",si);
getch();
}
```

##### OUTPUT #####

Simple interest is 500.000000

Note: variables of lower datatypes can be typecasted into higher order datatypes but vice versa is not supported. Here higher datatype to lower datatype typecasting is not supported because higher datatype variable covers more bytes in memory, and that value can't be accommodated into less memory occupied by lower datatype variable.

For example, a float variable cannot be typecasted into integer datatype because float covers 4 bytes in memory & integer covers 2 bytes only, so how 'C' can support data loss while casting.

## Summary

- 'C' Language Was Developed by Dennis Ritchie of AT & T Bell Laboratories in 1972.
- Turbo C is an integrated development environment & compiler developed by Borland to program in C

## 'C' Scope

Language. Turbo C version 1.0 was out in the market in 1987.

- Character set in 'C' denotes all the characters that have some valid meaning in 'C'
- The smallest individual words with or without symbols that contains some particular meaning in C language are called C Tokens.
- Variables refer to memory location that contains data value.
- The words that have predefined meaning in 'C' are known as key words also called as reserved words
- Escape sequence is to escape the normal meaning of a character and is used for non printable characters in a program.
- Typcasting is simply a mechanism by which one can change the data type of a variable, no matter how it was originally defined.

## Self Review

Q1. Write C Expressions corresponding to following arithmetic expressions

- a.  $\frac{5a + 7b}{a - 2.5}$
- b.  $9x^5 + 4x^4 - 7x^3 + 2x^2 - 8x + 12$
- c.  $\frac{4x+b}{c} + \frac{9y-3a}{3f} - \frac{2e}{bf}$

Q2. Evaluate the expressions

- a. int a=4 , b=5 , c=8;  
a += b++ / c%b;  
a \* = c++\* a-- + --b;  
b% = ++b \* c/2;



## 'C' Scope

- b. `int j , k , l;`  
`j = ( k=9 , l=3 ,k+1 );`  
`k = ++j / 9 * 1 % 2 + k--;`  
`l*=-j/k+j*k--;`

Q3. Program to convert

- a. Celcius into farenhiet ( Hint  $c/5f-32/9$  )
- b. Kilogram into pounds ( Hint  $0.45359Kg = 1$  Pound)
- c. Yards into miles ( Hint  $1 \text{ mile} = 1760$  Yards)
- d. Miles into Kilo metre ( Hint  $1\text{mile} = 1.60934Km$ )

Q4. Program to calculate circumference & area of a circle & a rectangle. ( Given length & breadth of rectangle & radius of circle.)

Q5. Program to calculate surface area and volume of

- a. Cuboid
- b. Cube
- c. Cylinder
- d. Sphere
- e. Cone

# **CHAPTER 2**

## **INPUT/OUTPUT STATEMENTS IN 'C'**

Input and Output Statement can be categorized in two categories

1. Unformatted I/O Functions
2. Formatted I/O Functions

### **Unformatted I/O**

#### **Character based I/O**

For character based input, the available functions are `getchar()`, `getche()`, `getch()` and for output the available functions are `putchar()`, `putch()`

#### **`getchar();`**

`getchar ()` is a macro that reads a single character from a standard input device i.e. keyboard.

The syntax is  
`ch=getchar();`

## 'C' Scope

```
#include <stdio.h>
main()
{
    char ch;
    printf ("Enter a character : ");
    ch=getchar();
    putchar(ch);
}
```

##### OUTPUT #####

```
Enter a character : m↵
m
Enter any character : raju↵
r
```

Note : It accepts only a single character as in above example when we type raju it accepts only r which is first character

### **getche()**

It reads a single character given from the keyboard and echoes to the current text window and there is no need to press enter.

```
#include <stdio.h>
main()
{
    char ch;
    printf ("Enter a character : ");
    ch=getche();
    putchar(ch);
}
```

## 'C' Scope

##### OUTPUT #####

Enter a character : m

m

Enter a character : b

b

### **getch()**

It reads a single character from keyboard without echoing (display) it to the screen.

Syntax :

ch =getch();

Example

```
#include <stdio.h>
main()
{
    char ch;
    printf ("Enter a character : ");
    ch=getch();
    printf ("\n");
    putchar(ch);
}
```

##### OUTPUT #####

Enter a character :

M

Enter a character :

B

## 'C' Scope

In above example as we press any key it accepts the character without displaying the entered character.

**getc();**

It is a macro which is used to read a character from file as well as from standard input device

```
ch=getc(stdin);
```

```
main()
{
  char ch;
  printf ("Enter a character : ");
  ch=getc(stdin);
  printf ("\n");
  putchar(ch);
}
```

##### OUTPUT #####

Enter any character : m↵

M

Enter any character : b↵

B

## **Character Based Output**

**putchar()**

putchar() is a macro that outputs a character to the standard output device i.e. Monitor

## 'C' Scope

Syntax :

```
putchar(Variable);
```

Example:

```
main()
{
  char ch;
  printf ("Enter a character : ");
  ch=getc(stdin);
  printf ("\n");
  putchar(ch);
}
```

##### OUTPUT #####

Enter any character : m↵

m

Enter any character : b↵

b

**putch();**

Outputs character to the text window on the screen and it does not translate linefeed character(\n) into carriage-return/linefeed combination

Syntax :

```
putch(Variable);
```

Example:

```
main()
{
  char ch;
```

## 'C' Scope

```
printf("Enter a character : ");
ch=getc(stdin);
printf("\n");
putch(ch);
}
```

Enter any character : m↵

m

Enter any character : b↵

b

Here in above program the output is same as the putchar example but while using for a file putchar convert the newline(\n) character into carriage-return and linefeed combination, whereas putch() does not convert the same. It treat the newline as a newline only.

Let us see the output of the following program using putchar() and putch() separately

```
#include <stdio.h>
main()
{
FILE *fp;
char ch;
clrscr();
fp=fopen("scope.txt","r");
ch=getc(fp);
while (ch !=EOF)
{
putchar(ch);
ch=getc(fp);
}
}
```

## 'C' Scope

```
getch();  
}
```

##### OUTPUT #####

This is a book.  
It is published by Scope.  
It was written by Nishat.  
It is based on the 'C' Fundamentals.

```
#include <stdio.h>  
main()  
{  
FILE *fp;  
char ch;  
clrscr();  
fp=fopen("scope.txt","r");  
ch=getc(fp);  
while (ch !=EOF)  
{  
putch(ch);  
ch=getc(fp);  
}  
getch();  
}
```

##### OUTPUT #####

This is a book.  
It is published by Scope.  
It is written by Nishat.  
  
It is based on the 'C' Fundamentals.



## 'C' Scope

### **putc()**

It is a macro which sends the output to a stream

Syntax :

```
putc(variable,stream);
```

Example :

```
#include <stdio.h>
main()
{
char ch;
clrscr();
printf ("Enter any character : ");
ch=getchar();
putc(ch,stdout);
}
```

##### OUTPUT #####

```
Enter any character : t↵
t
```

### **printf ()**

We can also print the value of a character using printf

Syntax :

```
printf ("%c",variable);
```

## 'C' Scope

### String Based I/O

#### gets()

Waits for user response and accepts a string of characters terminated by newline and given from the keyboard. It replaces the newline character by a character '\0' called as NULL. Inside the string it also allows the whitespaces.

Syntax :

```
gets(str)
```

Example :

```
#include <stdio.h>
main()
{
    char str[50];
    printf ("Enter a string of characters : ");
    gets(str);
    printf ("\n The given string is : %s ",str);
}
```

##### OUTPUT #####

```
Enter a string of characters: Scope Computer Education.↵
The given string is: Scope Computer Education
```

#### Accepting a string using scanf ()

Syntax :

```
Scanf ("%s",str);
```

## 'C' Scope

Example :

```
#include <stdio.h>
main()
{
    char str[50];
    printf ("Enter a string : ");
    scanf ("%s",str);
    printf("The given string is : %s ",str);
}
```

##### OUTPUT #####

Enter the string : Scope Computer Education  
The given string is : Scope

When we use scanf() to accept a string it does not allow the whitespaces inside the string so the output would be **Scope** only.

### **String Based Output:**

**puts()**

It gives the output of a string to standard output device i.e. Monitor, and also appends a new line character at the end of a string.

Syntax :  
puts(str);

Example :

```
#include <stdio.h>
main()
```

## 'C' Scope

```
{
char str[50],str1[50];
printf ("Enter a string :");
gets(str);
printf ("Enter the second string : ");
gets(str1);
puts(str);
puts(str1);
}
```

##### OUTPUT #####

```
Enter a string : Scope Computer Education↵
Enter the second string : Jodhpur Rajasthan↵
Scope Computer Education
Jodhpur Rajasthan
```

### **Output of a string using printf()**

While printing the output of a string using printf() it sends the output to standard output device Monitor but unlike puts() it does not add a new line character at the end of a string.

```
Enter a string: Scope Computer Education↵
Enter the second string: Jodhpur Rajasthan↵
Scope Computer Education Jodhpur Rajasthan
```

### **Formatted Input using scanf() Function**

Formatted input means an input data arranged format. The field or format specifier can contain an optional weight.

## 'C' Scope

This field width specifier is optional i.e no restriction to use these kind of specifier but by employing them we can easily take the input required by our program.

The field specification can be given as

% w format\_string

Let's see an example,

```
#include<stdio.h>
main()
{
    int a,b,c;
    clrscr();
    printf("\n Enter the nine digit number \n");
    scanf("%3d %3d %3d",&a,&b,&c);
    printf("\n Given number is \n");
    printf("%d-%d-%d",a,b,c);
    getch();
}
```

##### OUTPUT #####

Enter the nine digit number

1213456789

Given number is

121-345-678

### **Formatted output using printf() function**

Likewise formatted input formatted output can be also be shown according need of the user.

## 'C' Scope

Let us see a code regarding this

```
#include<stdio.h>
void main()
{
    float a,b,c;
    clrscr();
    printf("\n Enter the number \n");
    scanf("%f %f %f",&a,&b,&c);
    printf("\n Given number is \n");
    printf("%0.2f-%0.5f-%3.2f",a,b,c);
    getch();
}
```

##### OUTPUT #####

Enter the number  
789.3456  
5123.3426  
55555.689

Given number is  
789.35-5123.34277-55555.69

## Summary

- For character based input, the available functions are getchar(), getche(), getch() ,gets().
- For output the available functions are putchar(), putch() , puts() .

## 'C' Scope

- There are also other functions like printf() and scanf() for printing/Displaying the output or message on the execution screen and for taking the input from user.

## Self Review

- Q1. What is the difference between getch() and getche()?
- Q2. What is the difference between putch() and putche() ?
- Q3. What do you mean by formatted input and output.  
Explain with the help of example ?

# CHAPTER 3

## CONDITIONAL CONSTRUCTS

Students if you want to execute a statement or a set of statements only when certain conditions are true or to execute some other statements when the same conditions are false.

There are five constructs used in 'C' for employing these conditions.

1. if
2. if... else...
3. if ... elseif ... elseif ..... else
4. Nested if
5. switch ... case

### 1. if

Suppose if you want to execute a part of program if a particular condition is true then we can enclose these statements within a pair of { }.

```
if (condition)
{
    Statement1;
    Statement 2;
    .
    .
    .
    Statement n;
}
```



## 'C' Scope

Let us see an example,

```
#include<stdio.h>
main()
{
    int i=10;
    if(i > 5 )
        printf ("JACK");
    getch();
}
```

#####OUTPUT#####

JACK

In the program below, segment it will not print anything because first the value of i will be initialized with 10 and then it will check the condition here i is not less than 5 so it will not go to the part enclosed in the pair of curly bracket.

NOTE: Inside the true block if there is only one statement to be executed then we don't have the need to put curly braces ( { } ).

```
#include<stdio.h>
main()
{
    int i=10;
    if(i < 5 )
        printf ("JACK");
    getch();
}
```

## 'C' Scope

```
#####OUTPUT#####
```

None

### ii. if...else...

In the program explained in previous section the output was JACK what if the condition results false, the output was nothing. If we want certain statements to be executed based on false condition then use else extension of if construct.

```
if (condition)
{
    Statement 1;
    Statement 2;
    .
    .
    .
    Statement n;
}
else
{
    Statement 1;
    Statement 2;
    .
    .
    .
    Statement n;
}
```

Let's see an example

## 'C' Scope

```
#include<stdio.h>
main()
{
    int i=10;
    if(i < 5 )
    {
        printf ("JACK");
    }
    else
    {
        printf ("JILL");
    }
    getch();
}
```

Run part of above program

JILL

The above program segment will print the JILL because given condition is not true so the control will enter to the else part of the program and hence JILL will be printed.

So in another way we can say the control enter the first part of the program if the given condition is true otherwise it goes to the else part of the program.

Let us see another program segment

Ex. 1.

Calculate the net income of an employee, where employee can be clerk or other, if clerk his ta,da, hra will be 3,3.5,4 respectively, and for others it is 4,4.5,5 respectively

## 'C' Scope

```
#include<stdio.h>
void main()
{
    float salary,ta,da,hra,tot_income;
    char desig;
    clrscr();
    printf("\n Enter the salary of a person : \t");
    scanf("%f",&salary);
    fflush(stdin);
    printf("\n Enter 'C' for Clerk and other for others :
\t");
    scanf("%c",&desig);
    if(((desig=='C')||(desig=='c'))
    {
        ta=(3.0*salary)/100;
        da=(3.5*salary)/100;
        hra=(4.0*salary)/100;
        tot_income=salary+ta+da+hra;
    }
    else
    {
        ta=(4.0*salary)/100;
        da=(4.5*salary)/100;
        hra=(5.0*salary)/100;
        tot_income=salary+ta+da+hra;
    }
    printf("The total income for you is : \t
%f",tot_income);
    getch();
}
```

#####OUTPUT#####

## 'C' Scope

Enter the salary of a person : 10000.00  
Enter 'C' for Clerk and other for others : c  
The total income for you is : 11050.000000

### iii. if.... elseif.... elseif.... else...

If we have more than one condition to check then we can use if ... elseif ... elseif... else construct.

Ex. 2.

Print the Grade of a student based on percentage

```
#include<stdio.h>
main()
{
    int tmarks;
    printf ("Enter total marks : ");
    scanf ("%d",&tmarks);
    if ( tmarks >= 60 )
    {
        printf ("First Division : ");
    }
    else if (tmarks >=48 )
    {
        printf (" Second Division :");
    }
    else if (tmarks >= 36 )
    {
        printf ("Third Division :");
    }
    else
```

## 'C' Scope

```
{  
    printf ("Failed : ");  
}  
getch();  
}
```

#####OUTPUT#####

```
Enter total marks : 65↵  
First Division :  
Enter total marks : 56↵  
Second Division :  
Enter total marks : 37↵  
Third Division :  
Enter total marks : 25↵  
Failed :
```

Above program segment will print the division according to the total marks. i.e its first condition will be checked if it is true then First division will be printed and if first condition is false then second condition written with else if will be checked if it is true then Second Division will be printed and if second condition is false then the third condition written with next elseif and so on....

If it finds all conditions as false then else block will be executed.

This construct is also known as if – else – elseif - else ladder.

### **iv. Nested if construct**

In else – if ladder we can only put conditions again and again inside else block. But what if too many conditions are

## 'C' Scope

to be satisfied for single block of statements to be executed. Here in this construct we can nest the conditions within some conditions.

Let us see an example for this construct,

Ex. 3.

Print the Largest out of three numbers

```
#include<stdio.h>
```

```
void main()
{
    int a,b,c;
    clrscr();
    printf("\n Enter the three numbers \n");
    scanf("%d",&a);
    scanf("%d",&b);
    scanf("%d",&c);
    if(a>b)
    {
        if(a>c)
            printf("\n a is largest");
        else
            printf("\n c is largest");
    }
    else
    {
        if(b>c)
            printf("\n b is largest");
        else
            printf("\n c is largest");
    }
}
```

## 'C' Scope

```
    getch();  
}
```

##### OUTPUT #####

Enter the three numbers

10

11

16

c is largest

Enter the three numbers

10

12

8

b is largest

Enter the three numbers

12

8

10

a is largest

### v. **Switch statement**

It is the multiway decision construct and is the alternative statement for if .. else if .. else construct.

It tests the value of a given variable or expression and switch over to the specific case, there it executes all the statement with in the boundary of switch till it does not get the break; statement.



## 'C' Scope

If it does not match the value in any case then control goes to the default case.

The syntax of the switch statement is :-

```
switch(variable)
{
case value1 :
    ----
    ----
    break;
case value2 :
    ----
    ----
    break;
case value 3 :
    ----
    ----
.
.
case value n :
    ----
    ----
    break;
default :
    ----
    ----
}
```

The break statement is here used to jump out of the switch whenever any case is matched against value of switch variable so that the compiler time is not wasted roaming other statements unnecessarily.

## 'C' Scope

First let's see the following example using if.. else if ..else construct

```
#include<stdio.h>
main()
{
    int i;
    printf ("Enter the value of i ");
    scanf ("%d",&i);
    if (i==1)
        printf ("SCOPE");
    else if (i==2)
        printf ("NSD");
    else
        printf ("Out of range .");
    getch();
}
```

here is the output

```
Enter the value of i : 1
SCOPE
Enter the value of i : 2
NSD
Enter the value of i : 4
Out of range
```

The above program can be written using switch statement which is more convenient and easy to understand.

```
#include <stdio.h>
main()
```

## 'C' Scope

```
{
int i;
printf ("Enter a number : ");
scanf ("%d",&i);
switch(i)
{
case 1 :
    printf ("Scope\n");
    break;
case 2 :
    printf ("NSD\n");
    break;
default :
    printf ("Out of range : \n");
}
getch();
}
```

##### OUTPUT #####

```
Enter a number : 1
Scope
Enter a number : 2
NSD
Enter a number : 4
Out of range :
```

In above example as we enter the value of i as 2 the control switch over to the second case and it print the NSD and when we enter the value as 4 the control goes to the default case. It can be considered as the else part of if .. else if ... else construct.

## 'C' Scope

Ex. 4.

Program to demonstrate switch statement

```
#include <stdio.h>
main()
{
    int i;
    printf ("Enter a number : ");
    scanf("%d",&i);
    switch(i)
    {
        case 1 :
            printf ("Scope\n");
        case 2 :
            printf ("NSD\n");
        default :
            printf ("Out of range : \n");
    }
    getch();
}
```

##### OUTPUT #####

```
Enter a number : 1
Scope
NSD
Out of range :
Enter a number : 2
NSD
Out of range :
Enter a number : 3
Out of range :
```

## 'C' Scope

---

In above program we have not used break; statement, so when we use the value of i as 1 then control passes through all the statement of all cases including the default case. When we use the value of i as 2 then the control switches over to the second case and from second case it passes through all the statement of switch. This property of falling through till it does not get the break statement is known as *fall through property*.

## Summary

There are five constructs used in 'C' for employing these conditions, if, if... else..., if ... elseif ... elseif ..... else, Nested if, switch ... case.

If you want to execute a part of program if a particular condition is true then you can use a simple if construct.

If we want certain statements to be executed based on false condition then use else extension of if construct.

If we have more than one condition to check then we can use if ... elseif ... elseif... else construct. It is also called as If – else – ladder because the programme flow appearance looks like ladder case.

If too many conditions are to be satisfied for single block of statements to be executed, we can nest the conditions within if - else conditions. This is also called as Nested If construct.

## 'C' Scope

Switch is the multiway decision construct and is the alternative statement for if .. else if .. else construct. It checks for the variable or expression in switch() for tautology

### Self Review

- Q1. Write a program to detect whether the given number is odd or even?
- Q2. Write a program to find whether the given integer is positive or negative?
- Q3. Write a program to find largest among 2 integers, three integers, 4 integers?
- Q4. Write a program to find whether the given year is a leap year or not?
- Q5. Write a program to find today is which day of the year?
- Q6. Write a program to find whether the given angles of triangle are valid or not( Hint: sum of three angles of triangle is 180)
- Q7. Write a program to check which key is pressed of keyboard, whether it is digit,symbol, capital or small alphabet?
- Q8. Write a program to calculate Electricity if following is the criteria

Units consumed	Rs. Per unit
Below 100	1 Rs.
101 – 300	1.5 Rs.
301 – 500	2.0 Rs.
501 – 1000	3.0 Rs.
1000 & Above	3.5 Rs.

## 'C' Scope

Also the compulsory charges according to the category i.e for domestic its 180 Rs., for commercial it is 360 Rs.?

- Q9. Write a program to check whether the given point  $p(x,y)$  is in which quadrant?
- Q10. Write a program that inputs time of the day & prints Greeting for the user accordingly ?
- Q11. Write a program to find that today is which day of the year i.e for example 15<sup>th</sup> Feb is 46<sup>th</sup> day of the year?
- Q12. Write a program to find difference between two dates ?
- Q13. Write a program to compute the TDS deducted from salary of employee if the salary is taken by the user and the tax percentage depends on income tax slab system ?
- Q14. Write a program to find out the grade of student based on his marks in 5 subjects;
- |               |         |
|---------------|---------|
| 80% and above | A Grade |
| 60% - 79%     | B Grade |
| 40% - 59%     | C Grade |
| 39% or Below  | D Grade |

# CHAPTER 4

## LOOPS

Loops are the constructs used to execute the part of a program repeatedly.

There are three types of loops in ‘C’

- i. for loop
- ii. while loop
- iii. do .. while loop

### i. for loop

In for loop first we give the initial value and then it checks the condition, if the condition is true then it enters the loop and execute all the statement inside the curly brackets and after this it goes to the third part that is changing variable value and it again checks the conditions. In this way it runs till the condition is true.

When the condition becomes false it comes out of the loop.

The general form of for loop is

```
for(initial value; condition check; re initialization part)
{
    -----
    body of the loop
    -----
}
```



## 'C' Scope

```
for(initialization; terminating condition; increment/ decrement step)
{
    -----
    Body of the loop
    -----
}
```

```
for (i=0;i<10;i++)
{
    printf (“\n%d”,i);
}
```

steps for this loop is as follows

1.  $i=0$  this will set the initial value of  $i$  to zero.
2. control will transfer to the second part  $i<10$  and check the condition whether it is true or false. This time condition is true. so the control will be entered to the body of the loop.
3. statement `printf (“\n%d”,i);` will be executed and the value of  $i$  will be printed first it will print 0.
4. after the execution of all the statements of the body of the loop control will transfer to increment/decrement part  $i++$  the value of  $i$  will be incremented it will become 1.
5. now again the transfer of control goes for checking the termination condition, if it is true enters the body of the loop, executes the statements written there, if it is false the transfer of control jumps to the next statement given after the closing brace of loop.
6. In this way in the program the loop will work from  $i=0$  till  $i<10$  i.e.  $i=9$  & prints numbers from 0-9.

## 'C' Scope

This program segment will first set the initial value of *i* to 0 and then it will check the condition which is true this time (0 is less than 10), so it will enter into the body of the loop where it will print the value of *i* i.e. 0 (execute all the statement inside the body of the loop) then it will go to the reinitialization part and increase the value of *i* to 1, and then after this it will again check the condition it is true again so it will reenters to the body of the loop and repeat the same procedure till the condition is true.

The output of the above segment would be

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

Ex. 1.

Program to print the temperature conversion table that shows fahrenheit & centigrade values.

```
#include <stdio.h>
main()
{
    int fah;
    for (fah=0;fah <=20;fah++)
```

## 'C' Scope

```
printf("%dfahrenheit=%fcentigrade\n ",fah,(5.0/9.0)*(fah-32));  
}
```

Here is the output

```
0 fahrenheit = -17.777778 centigrade  
1 fahrenheit = -17.222222 centigrade  
2 fahrenheit = -16.666667 centigrade  
3 fahrenheit = -16.111111 centigrade  
4 fahrenheit = -15.555556 centigrade  
5 fahrenheit = -15.000000 centigrade  
6 fahrenheit = -14.444444 centigrade  
7 fahrenheit = -13.888889 centigrade  
8 fahrenheit = -13.333333 centigrade  
9 fahrenheit = -12.777778 centigrade  
10 fahrenheit = -12.222222 centigrade  
11 fahrenheit = -11.666667 centigrade  
12 fahrenheit = -11.111111 centigrade  
13 fahrenheit = -10.555556 centigrade  
14 fahrenheit = -10.000000 centigrade  
15 fahrenheit = -9.444444 centigrade  
16 fahrenheit = -8.888889 centigrade  
17 fahrenheit = -8.333333 centigrade  
18 fahrenheit = -7.777778 centigrade  
19 fahrenheit = -7.222222 centigrade  
20 fahrenheit = -6.666667 centigrade
```

### **Class room Question**

# What happens if we miss initialization or reinitialization part in a loop?

## 'C' Scope

⇒ We can negate the initialization or reinitialization part of the loop

let us see how,

```
#include <stdio.h>
main()
{
  int i=0;
  for( ; i < 10 ; )
  {
    printf ("%d",i);
    i++;
  }
}
```

In above program i is initialized with 0 at the time of declaration so it is not necessary to initialize it again in construction of 'for loop', and reinitialization is written inside the body of the loop. So it is again not necessary to write it in for loop construct.

What happens if we neglect the condition in for loop construction?

⇒ Condition part of a loop is necessary which actually decides that it will execute or not, if we negate the condition part it will execute INFINITELY.

```
#include <stdio.h>
main()
{
  int i=1;
```

## 'C' Scope

```
clrscr();
for (;;)
{
    printf ("%d",i);
    i++;
}
getch();
}
```

Above program will run infinitely.

#What will be the effect if we terminate the for construction with semicolon (;) ?

⇒ For the answer of this question let's see the output of this program

```
#include <stdio.h>
main()
{
    int i;
    for(i=0;i<10;i++);
    {
        printf ("%d",i);
    }
    and here is the output
    10
```

In the given program there is termination sign(;) in for construct so it will not allow control to go to the body when the condition is true, it will force the control to transfer to the reinitialization part after checking the true condition. It repeats within the for construct till the condition is true.

## 'C' Scope

When the condition becomes false it goes to the printf statement, so the output would be 10 only.

Ex. 2.

Program To print the table of a given number

```
#include <stdio.h>
main()
{
    int num,i;
    clrscr();
    printf("Enter a number : ");
    scanf ("%d",&num);
    fflush(stdin);
    for (i=1;i<=10;i++)
    {
        printf ("\n%d X %d = %d",num,i,num*i);
    }
    getch();
}
```

The output of the program would be

Enter a number : 8 ↵

8 X 1 = 8

8 X 2 = 16

8 X 3 = 24

.

.

8 X 9 = 72

8 X 10 = 80

Suppose we want to print the range of integer

## 'C' Scope

1,2,3..... 32767, -32768,-32767,-32766.....-2,-1,0

Let's check out the output of the following program.

Ex. 3.

Program to print the range of integer

```
#include <stdio.h>
main()
{
    int i;
    clrscr();
    for (i=0;i<32768;i++)
    {
        printf (" %d",i);
    }
    getch();
}
```

The result will be an infinite loop

As 0,1,2...3767,-32868,-32767... 0,1,2...

because the maximum range of the integer variable is 32767 when we try to exceed by 1 it will be -32768 and in this way it will go to 0 and try to increase the values infinitely.

### **Nesting of for loops**

Like Nested -If construct we can also nest for loops i.e put loop inside loops.

The general body of nested loop can be explained by the following syntax.

## 'C' Scope

```
for(initialize;termination condition;increment/decrement)
{
    -----
    Body of outer loop
    -----

    for(initialize;termination condition; incr/decr)
    {
        -----
        Body of inner loop
        -----
    }
}
```

Let us see an example

```
for(i=0;i<10;i++)
{
    for(j=0;j<10;j++)
        printf(“%d %d”,i , j );
}
```

The above code would be executed one time to give 100 output lines. For one value of i the whole inner loop of j would execute, & for the next value of i the inner loop of j would execute fully once again and so on. The output of the above program would be.

```
0    0
0    1
0    2
0    3
```



## 'C' Scope

```
|
|
0 9
1 0
1 1
1 2
|
|
9 6
9 7
9 8
9 9
```

Ex. 4.

```
*
**
***
****
*****
```

To print a triangle pattern according to number of rows

```
#include<stdio.h>
main()
{
int i,rows,j;
clrscr();
printf ("Enter Number of rows :");
scanf ("%d",&rows);
for(i=0;i<rows;i++)
```

## 'C' Scope

```
{
for(j=0;j<=i;j++)
{
printf ("*");
}
printf ("\n");
}
getch();
}
```

Run part of above program

Enter number of rows :5↵

```
*
**
***
****
*****
```

Ex. 5.

Print the star triangle in the mid of the screen like below

```
      *
     **
    ***
   ****
  *****
```

```
#include <stdio.h>
main()
{
int x,y,z,sp=1,middle=40;
clrscr();
printf("Enter the number:=");
scanf("%d",&z);
```

## 'C' Scope

```
for(y=1;y<=z;y++)
{
for(sp =1;sp<=middle;sp++)
printf(" ");
middle--;
for(x=1;x<=y;x++)
printf("* ");
printf("\n");
}
getch();
}
```

Run part of above program:= 5

```
      *
     **
    ***
   ****
  *****
```

Ex. 5.

Print the triangle of stars in reverse order in middle of screen

```
*****
****
***
**
*
```

```
#include <stdio.h>
main()
{
int x,y,z,sp,middle=40;
clrscr();
```

## 'C' Scope

```
printf("Enter the number:=");
scanf("%d",&z);
for(;z>=1;z--)
{
for(sp=1;sp<=middle;sp++)
printf(" ");
middle++;
for(y=z;y>=1;y--)
printf("* ");
printf("\n");
}
getch();
}
```

Run part of above program

Enter the number:=5.↵

```
* * * * *
* * * * *
* * *
* *
*
```

Ex. 6.

Program to print Rhombus of a number (1)

```
1
1 2 1
1 2 3 2 1
1 2 1
1
```

```
#include<stdio.h>
main()
{
int x,y,z,sp=1,middle=40;
clrscr();
printf("Enter the number:=");
```

## 'C' Scope

```
scanf("%d",&z);
for(y=1;y<=z;y++)
{
    for(sp=1;sp<=middle;sp++)
        printf(" ");
    middle-=2;
    for(x=1;x<=y;x++)
        printf("%d ",x);
    for(x-=2;x>=1;x--)
        printf("%d ",x);
    printf("\n");
}
middle+=4;
for(z--;z>=1;z--)
{
    for(sp =1;sp<=middle;sp++)
        printf(" ");
    middle+=2;
    for(x=1;x<=z;x++)
        printf("%d ",x);
    for(x-=2;x>=1;x--)
        printf("%d ",x);
    printf("\n");
}
getch();
}
```

Run part of above program :=

Enter the number:=5.↓

```
1
1 2 1
1 2 3 2 1
1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
```

## 'C' Scope

```
1 2 3 4 3 2 1
  1 2 3 2 1
    1 2 1
      1
```

Ex. 7.

Program to print the rhombus of a number like below

```
5
 4 5 4
3 4 5 4 3
2 3 4 5 4 3 2
1 2 3 4 5 4 3 2 1
 2 3 4 5 4 3 2
   3 4 5 4 3
    4 5 4
     5
```

```
#include<stdio.h>
main()
{
  int x,y,z,sp=1,middle=40;
  clrscr();
  printf("Enter the number:=");
  scanf("%d",&z);
  for(y=z;y>=1;y--)
  {
    for(sp=1;sp<=middle;sp++)
      printf(" ");
    middle-=2;
    for(x=y;x<=z;x++)
      printf("%d ",x);
    for(x-=2;x>=y;x--)
      printf("%d ",x);
    printf("\n");
  }
  middle+=4;
```

## 'C' Scope

```
for(y=2;y<=z;y++)
{
    for(sp=1;sp<=middle;sp++)
        printf(" ");
    middle+=2;
    for(x=y;x<=z;x++)
        printf("%d ",x);
    for(x-=2;x>=y;x--)
        printf("%d ",x);
    printf("\n");
}
getch();
}
```

Run part of above program

Enter the number:=5.↵

```
      5
     4 5 4
    3 4 5 4 3
   2 3 4 5 4 3 2
  1 2 3 4 5 4 3 2 1
 2 3 4 5 4 3 2
 3 4 5 4 3
 4 5 4
 5
```

Ex. 8.

Program to print the floyd triangle

```
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
```

```
#include<stdio.h>
main()
```

## 'C' Scope

```
{
int num,y,z;
clrscr();
printf("Enter the last row number:=");
scanf ("%d",&num);
fflush(stdin);
for(y=1;y<=num;y++)
{
    if(y%2 != 0)
    {
        for(z=1;z<=y;z++)
        {
            if(z%2 !=0)
                printf("1 ");
            else
                printf("0 ");
        }
        printf("\n");
    }
    else
    {
        for(z=1;z<=y;z++)
        {
            if(z%2 !=0)
                printf("0 ");
            else
                printf("1 ");
        }
        printf("\n");
    }
}
getch();
}
```



## 'C' Scope

Run part of above program

Enter the last row number:=5↵

```
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
```

Ex. 9.

Program to print the Pascal triangle

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

```
#include<stdio.h>
main()
{
    int c,r,row,bc;
    clrscr();
    printf("Enter the last row number:=");
    scanf ("%d",&row);
    fflush(stdin);
    for(r=0;r<row;r++)
    {
        for(bc=1,c=0;c<=r;c++)
        {
            if(c==0)
                printf("\n%-3d",bc);
            else
```

## 'C' Scope

```
{
    bc=bc *(r-c+1)/c;
    printf("%-3d",bc);
}
}
}
getch();
}
```

Run part of above program

Enter the last row number:=7↵

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
```

Ex. 10.

Program to print the Binomial Coefficient

```
#include <stdio.h>
main()
{
    int ncr,n,r,a,b,x;
    clrscr();
    printf ("Enter the value for n : ");
    scanf ("%d",&n);
    printf ("Enter the value for r : ");
    scanf ("%d",&r);
```

## 'C' Scope

---

```
r > (n-r) ? (a=r,b=n-r) :( a=n-r,b=r);
for (x=n;x > a;x--)
    ncr = ncr * x;
for (x=2;x <=b;x++)
    ncr= ncr/ x;
printf ("The value of ncr is %d ",ncr);
getch();
}
```

The output of the following program would be

Enter the value for n : 5  
Enter the value for r : 10  
The value of ncr is 1316

Enter the value for n : 60  
Enter the value for r : 1  
The value of ncr is 13424

Enter the value for n : 6  
Enter the value for r : 5  
The value of ncr is 7896

### ii. while loop

The most common form of while loop is

In while loop we first write the initial value and then we checks the condition if the condition is true it enters the loop otherwise it comes out. In true condition it executes all the statement inside the curly brackets. And then again checks the condition this process repeats till the condition

## 'C' Scope

is true. While loop is also called as *entry controlled loop construct* because it again & again checks the condition before entering & executing the body of the loop i.e. at the starting only.

```
initial value;
while(condition checking)
{
    -----
    body of the loop
    -----
}
```

Ex. 11.

Program to reverse a given number

given number => 123

output => 321

```
#include<stdio.h>
main()
{
    int num,result;
    clrscr();
    printf("Enter the number:=");
    scanf("%d",&num);
    fflush(stdin);
    while(num >=1)
    {
        result =num % 10;
        num=num/10;
        printf("%d",result);
    }
```

## 'C' Scope

```
getch();  
}
```

Run part of above program

Enter the number:=2735↵

5372

The program explained how to reverse a number using a simple while loop, here the decrement of counter is taken

Ex. 12.

Explain the position of individual digit in a number

```
#include<stdio.h>  
main()  
{  
int num,temp,div=1,store,total=0;  
clrscr();  
printf("Enter the number:=");  
scanf("%d",&num);  
total=temp =num;  
for(;num>=1;num=num/10)  
div*=10;  
div/=10;  
while(temp >=1)  
{  
store = temp/div;  
temp%=div;  
printf(" ( %d * %d ) +",store,div);  
div/=10;  
}  
}
```

## 'C' Scope

```
printf("\b= %d",total);  
getch();  
}
```

Run part of above program

Enter the number:=1437↵

$(1 * 1000) + (4 * 100) + (3 * 10) + (7 * 1) = 1437$

### iii. do .. while loop

do while loop construct will first execute all the statements written inside the loop and then it checks the condition if the condition is true it re-enters the loop, otherwise it comes out. So this loop will run at least once. That's why this loop is also called as *exit controlled looping construct*.

```
initial value;  
do  
{  
    -----  
    body of the loop  
    -----  
}while(condition checking);
```

Ex. 13.

Program to generate the Fibonacci series

```
#include<stdio.h>  
main()  
{  
    int fibo,store =0,value =1,result =0;
```

## 'C' Scope

---

```
clrscr();
printf("Enter the last number:=");
scanf ("%d",&fibonacci);
fflush(stdin);
do
{
printf("%d ",result);
result =value +store;
value =store;
store =result;
}
while(result <= fibonacci);
getch();
}
```

Run part of above program

Enter the last number:=100.↵

0 1 1 2 3 5 8 13 21 34 55 89

## Jump Statements

'C' employs jump statements to perform unconditional branch to a block or any particular statement in the program.

These jump statements are of four types.

- i) The return Statement
- ii) The goto Statement
- iii) The continue Statement
- iv) The break Statement

### **The return statement**

## 'C' Scope

The return statement is used to return from a function. It is categorized as jump statement because it causes execution to return (jump back) to the point at which the call to the function was made.

A return statement may or may not have a value associated with it. A return with a value can be used only in a function with a non-void return type. In this case, the value associated with return becomes the return value of the function. A return without a value is used to return from a void function.

The general syntax of this statement is  
return expression;

### **The goto Statement**

The goto statement in 'C' can be used to jump to any particular area in program, such as jumping out of set of deeply nested loops. This goto statement if used wisely can be a benefit in narrow set of programming situations.

The goto statement requires a label for operation. The general form of goto statement is

goto *label*;

.  
. .  
. .  
. .

*label*;



## 'C' Scope

Let us see an example,

```
X=1;
Loop1;
    X++;
    if(X<=100) goto Loop1;
```

### **The break Statement**

This break statement can be employed in two ways; the first use we have seen during switch case. And we can also use it to jump out of the loop i.e. to force immediate termination of loop, bypassing the normal loop termination conditional test.

```
#include<stdio.h>

void main()
{
    int t;
    clrscr();
    for(t=0;t<100;t++)
    {
        printf("%d",t);
        if(t==10) break;
    }
    printf("\n printed t till 10");
    getch();
}
```

OUTPUT:-

## 'C' Scope

---

012345678910

Printed t till 10

### Summary

- Loops are the constructs used to execute the part of a program repeatedly.
- In for loop first we give the initial value and then it checks the condition, if the condition is true then it enters the loop and execute all the statement inside the curly brackets and after this it goes to the third part that is changing variable value and it again checks the conditions.
- In while loop we first write the initial value and then we checks the condition if the condition is true it enters the loop otherwise it comes out. In true condition it executes all the statement inside the body of loop.
- do while loop construct will first execute all the statements written inside the loop and then it checks the condition if the condition is true it re-enters the loop, otherwise it comes out.
- Jump statements to perform unconditional branch to a block or any particular statement in the program. These are four types, namely return statement, goto statement, continue statement, break statement

### Self Review

Q1. Write the program to print the following patterns:-

a) \*

## 'C' Scope

```
  * *  
 * * *  
* * * *
```

b) 1  
1 2  
1 2 3  
1 2 3 4

c) 1 1  
12 21  
12321

d) ABCDEFGH  
 ABCDEFG  
 ABCDEF  
 ABCDE  
 ABCD  
 ABC  
 AB  
 A

e) \*  
 \*\*\*  
\*\*\*\*\*  
 \*\*\*  
 \*

f) \*  
 \*\*  
\*\*\*

## 'C' Scope

```
****
***
**
*
```

g) 1  
22  
333  
4444  
55555

h) \*\*\*\*\*  
\*\*\* \*\*\*  
\*\* \*\*  
\* \*

i) 1  
232  
34543  
4567654  
567898765

Q2. Write a program to print the tables from 2 to 15 ?  
Like the pattern below.

```
2 3 4 5 6 7 8 9 10 11 12 13 14 15
4 6 8 10 12 14 16 18 20 22 24 26 28 30
-
-
-
20 30 40 50 ..... 150
```

Q3. Write a program to find whether the given number is  
Armstrong number or not ?

## 'C' Scope

- Q4. Write a program to find factorial of a number ?
- Q5. Write a program to find first 10 armstrong numbers?
- Q6. Write a program to find whether the number is a prime number or not . Also find prime numbers between 1-100?
- Q7. Find the sum of sine & cos series upto 10 terms ?
- Q8. Write a program to compute compound interest for any amount , at a particular rate for time t using loops ?
- Q9. Write a program to fill the screen with smiley face until any key is pressed by the user?
- Q10. Write a program to generate Fibonacci series upto 10 elements?
- Q11. Write a program to convert decimal number to binary number?
- Q12. Write a program to perform sum of N natural numbers ?
- Q13. Write a program to find GCD and LCM of three natural numbers ?
- Q14. Write a program to check whether the given number is perfect, abundant or deficient ?

# CHAPTER – 5

## ARRAY

**In this chapter we will learn about the concept of array, how to use and create array.**

**We will also learn about**

One dimensional integer array  
One dimensional character array (string)  
Two dimensional integer array  
Two dimensional character array(List of strings)  
Concept of multidimensional array.

An Array is a collection of homogenous sequential memory locations named using single variable and differentiated using different indices.

### **One Dimensional Integer Array**

What is the meaning of the declaration?

```
int x = 10;
```

This declaration is the direction for compiler to :

- (i) Reserve 2 bytes of memory for an integer variable
- (ii) Associate the name x with memory address
- (iii) Store value 10 to that address

## 'C' Scope

9225	address of location	
<table border="1"><tr><td>10</td></tr></table>	10	value at address
10		
x	name of the location	

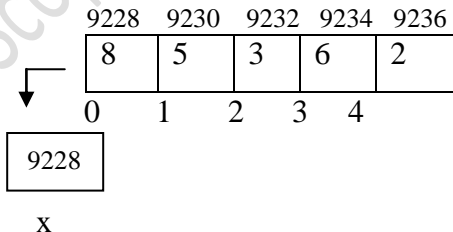
But if we want to store more than one numbers (list of numbers) with the same variable name then we have to use an array. The method of declaration of an array is in this way

**datatype name\_of\_array[number of elements in array];**

int x[5];

This declaration is the direction to compiler to:

- (i) reserve 10 contiguous bytes of memory for 5 different integers (because 1 integer occupies 2 bytes)
- (ii) associate the name x with the starting address of that memory location



## 'C' Scope

where the int shows the type of variable, x indicates the name of variable and [5] indicates how many elements of integers can be stored in our memory.

In the given figure the starting address is 9228 where the integer 8 is stored. i.e. 8 is stored at location no. 9228/9229 location (since integer occupies 2 bytes in the memory) in the same way next number is stored at the address 9230/9231 and so on.

[0],[1],[2].. are the indices or subscripts which are used to store, print or extract an individual integer.

To accept 5 integers we can use the following program segment.

Ex. 1.

Program to accept 5 integers

```
#include <stdio.h>
main()
{
    int num[5],i,j;
    //To Accept five different integers
    for (i=0;i<5;i++)
    {
        printf ("Enter any integers : ");
        scanf ("%d",&num[i]);
    }
    //To print the given integers
    for (i=0;i<5;i++)
    {
        printf ("\n%d",num[i]);
    }
}
```



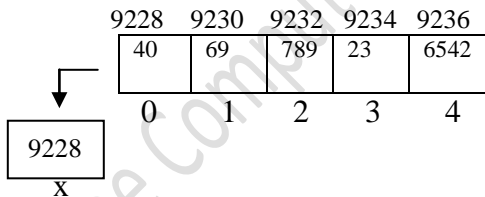
## 'C' Scope

```
}  
getch();  
}
```

##### OUTPUT #####

Enter an integers : 40  
Enter an integers : 69  
Enter an integers : 789  
Enter an integers : 23  
Enter an integers : 654

40  
69  
789  
23  
654



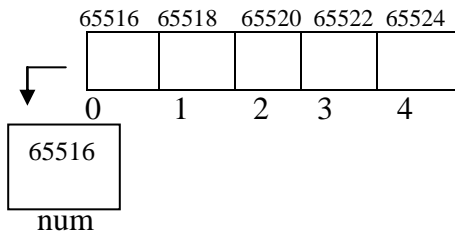
In above program we have entered five different integers the first integer 40 will be stored at num[0] i.e. at address 9228 and second integer 69 will be stored at num[1] at address 9230 and so on.

In the same manner we can print these numbers using their indices.

9228 is called as the base address of the array.

## 'C' Scope

& sign can be used to extract the address of the variable.  
If we want to print the starting address of the num then following statement can be used to print it.



```
printf ("%u",num);
```

Will print the starting address of array num and the address of num[0] which is also same as the value of num and which can be demonstrated by the following program segment.

```
#include <stdio.h>
main()
{
  int num[5];
  clrscr();
  printf ("\n\n%u",num);
  printf ("\n\n%u",&num[0]);
  getch();
}
```

```
#####OUTPUT#####
65516
```

## 'C' Scope

65516

We can also print the addresses of other locations using following program.

Ex. 2.

Program to print addresses of the locations, of an array

```
#include <stdio.h>
main()
{
    int num[5];
    int i,j,k;
    clrscr();
    for(i=0;i<5;i++)
    {
        printf ("Enter a number : ");
        scanf ("%d",&num[i]);
    }
    for(i=0;i<5;i++)
    {
        printf ("\n%d is stored at address %u",num[i],&num[i]);
    }
    getch();
}
```

\*\*\*\*output\*\*\*\*

Enter a number : 6

Enter a number : 5

Enter a number : 4

Enter a number : 3

Enter a number : 2

## 'C' Scope

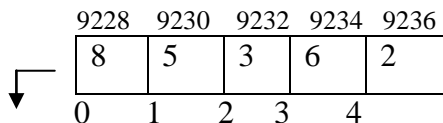
6 is stored at address 65516  
5 is stored at address 65518  
4 is stored at address 65520  
3 is stored at address 65522  
2 is stored at address 65524

Here are some examples where the concept of array can be used

- List of students in an institute.
- List of countries.
- Marks of examination for the students of a institute.
- Name of the employees of an organization and their salaries.
- List of any contents.

Technically we can say “ARRAY IS A SEQUENTIAL COLLECTION OF DATA OR LIST OF ITEMS SHARING A COMMON NAME”

See this memory segment



## 'C' Scope

9228
------

x

the type of data stored at x[0] is integer and x[1] is also integer. In the same way all the integers stored x[0],x[1],x[2],x[3] and x[4] have the same data type. So some times array is also called as homogeneous representation of memory, because all elements have the same datatype.

Array initialization

```
#include <stdio.h>
main()
{
    int x[5];
}
```

In this segment we have declared an array of 5 elements but value stored at these locations are the garbage values. So after declaring initialize the array with some meaningful data.

Array can be initialized in two ways

1. Initialization at the time of compilation.
2. Initialization of array at run time.

**How to initialize the integer array at compile time.**

## 'C' Scope

Look at this statement given below, in this statement we are initializing the array for the compile time by providing the list separated by the comma.

```
int x[5]={9,6,4,8,7};
```

this list must be initialized at the time of declaration so, if we will try to initialize the array in the following manner then there will be an error.

```
int x[5];  
x[5]={6,5,4,3,2};
```

### **How to initialize the integer array at run time.**

An integer array can be initialized at run time using scanf().

```
#include <stdio.h>  
main()  
{  
    int x[5];  
    int i;  
    for(i=0;i<5;i++)  
    {  
        scanf ("%d",&x[i]);  
    }  
}
```

In this example we are initializing the values for x[0],x[1]... at the run time using 'for' loop.

## 'C' Scope

We can also initialize array at run time using one by one initialization technique, as shown in following program segment.

```
#include <stdio.h>
main()
{
    int x[5];
    scanf ("%d %d %d %d %d", &x[0], &x[1], &x[2], &x[3], &x[4]);
}
```

Ex. 3.

Program to input ten numbers and print their sum

```
#include <stdio.h>
main()
{
    int num[10], i, sum;
    clrscr();
    for (i=0; i<10; i++)
    {
        printf ("Enter an integers : ");
        scanf ("%d", &num[i]);
    }

    for (i=0; i<10; i++)
    {
        sum=sum+num[i];
    }
    printf ("\n The sum of the given numbers is %d ", sum);
}
```

## 'C' Scope

```
getch();  
}
```

##### OUTPUT #####

```
Enter an integers : 23  
Enter an integers : 45  
Enter an integers : 6  
Enter an integers : 4  
Enter an integers : 36  
Enter an integers : 67  
Enter an integers : 2  
Enter an integers : 12  
Enter an integers : 11  
Enter an integers : 12
```

The sum of the given numbers is : 1516

Ex. 4.

Program to print the largest and smallest value of five numbers

```
#include <stdio.h>  
main()  
{  
    int i,j,k,num[5],max=0,min;  
    clrscr();  
    for (i=0;i<5;i++)  
    {  
        printf ("\nEnter a number : ");  
        scanf ("%d",&num[i]);  
    }  
}
```



## 'C' Scope

---

```
min=num[0];
for (i=0;i<5;i++)
{
if (max < num[i])
max=num[i];
if (min>num[i])
min=num[i];
}
printf ("\n The maximum number is %d ",max);
printf ("\n The minimum number is %d ",min);
getch();
}
```

##### OUTPUT #####

```
Enter a number : 9
Enter a number : 8
Enter a number : 7
Enter a number : 6
Enter a number : 5
```

```
The maximum number is 9
The minimum number is 5
```

### Selection Sort

Ex. 5.

Program to sort an array of 10 integers implementing selection sort technique

```
#include <stdio.h>
main()
{
int num[10],i,j,temp;
```

## 'C' Scope

```
clrscr();
for (i=0;i<10;i++)
{
printf ("Enter any number : ");
scanf ("%d",&num[i]);
}
for (i=0;i<10;i++)
{
for (j=i+1;j<10;j++)
{
if (num[i] > num[j])
{
temp=num[i];
num[i]=num[j];
num[j]=temp;
}
}
}
printf ("\nThe output of the sorted array is : ");
for (i=0;i<10;i++)
printf ("\n%d",num[i]);
getch();
}
```

##### OUTPUT #####

```
Enter any number : 6
Enter any number : 10
Enter any number : 9
Enter any number : 8
Enter any number : 4
Enter any number : 5
Enter any number : 3
```

## 'C' Scope

---

Enter any number : 2

Enter any number : 7

Enter any number : 1

The output of the sorted array is :

1

2

3

4

5

6

7

8

9

10

### Bubble Sort

Ex. 6.

Program to sort an array of 10 integers implementing bubble sort technique

```
#include <stdio.h>
main()
{
    int num[10],i,j,temp;
    clrscr();
    for (i=0;i<10;i++)
    {
        printf ("Enter any number : ");
        scanf ("%d",&num[i]);
    }
    for (i=0;i<10;i++)
    {
```

## 'C' Scope

```
for (j=0;j<10;j++)
{
    if (num[j] >= num[j+1])
    {
        temp=num[j];
        num[j]=num[j+1];
        num[j+1]=temp;
    }
}
for (i=0;i<10;i++)
    printf ("\n%d",num[i]);
getch();
}
```

##### OUTPUT #####

Enter any number : 6  
Enter any number : 10  
Enter any number : 9  
Enter any number : 8  
Enter any number : 4  
Enter any number : 5  
Enter any number : 3  
Enter any number : 2  
Enter any number : 7  
Enter any number : 1

The output of the sorted array is :

1  
2  
3  
4  
5

## 'C' Scope

6  
7  
8  
9  
10

### One – Dimensional Character Array

Char x;

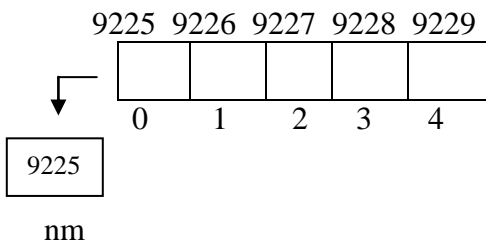
Consider the above declaration, this will occupy one byte of memory to store a single character for variable x.

But if we want to store the name of a student in variable x, then it is not possible because this variable is made to store a single character only.

If you want to store the name of a student then you have to occupy more bytes according to length of the name. For this purpose you have to use character array. A character array can be declared as shown below.

char nm[5];

In the same way we can use character array

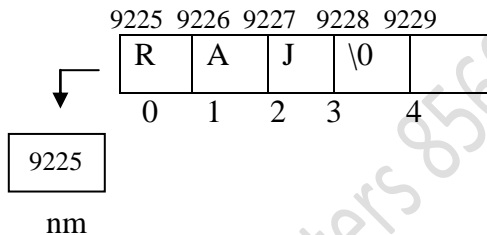


## 'C' Scope

The above declaration tells the compiler to occupy the 10 contiguous bytes of memory and associate the starting address (say 9225) of that memory with variable nm, where we can store a name of maximum 4 characters of name + one Null character for this variable.

An array can be initialized at the time of declaration like

```
char nm[5]="RAJ\0"
```



A null character will be produced at the end of the string RAJ. Individual character can be printed using the index of the character.

e.g.

```
printf ("%c",x[2]);
```

will print the character J because it is available at location number 2.

Where as in the following statement

```
printf ("%s",x);
```

## 'C' Scope

The control will go to the x and from there it will take the starting address of array and will start dumping of characters to the screen till it does not get the null.

And we see that the full string is printed i.e.  
RAJ

Character array can be initialized at the time of declaration only.

Try the following statements :-

```
char arr[10];  
arr="amit";
```

This will give an error because as we said earlier array can be initialize only at the time of declaration i.e.

```
char arr[10] = "amit" is a valid statement.
```

Array can also be initialized in the following manner

```
char arr[10] = {'a', 'm', 'i', 't', '\0'};
```

This is known as character by character initialization.

If we are initializing a character array at the time of declaration then it is not necessary to specify the size of array. If we do not specify the size of array at the time declaration the size of array will be determined automatically, according to the number of elements initialized.

## 'C' Scope

Eg.

```
char str[]="my angel";
```

here in this example variable will determine the size of array as 9.

```
#include<stdio.h>
main()
{
char str[]="My Angel";
clrscr();
printf ("\n\n%d",sizeof(str));
getch();
}
```

##### OUTPUT #####

9

Ex. 7.

Program to reverse a string

```
#include<stdio.h>
main()
{
char str[100];
int i,j,k;
clrscr();
printf ("Enter a string : ");
gets(str);
fflush(stdin);
```



## 'C' Scope

```
for(i=0;str[i];i++);
for(i--;i>=0;i--)
    printf ("%c",str[i]);
getch();
}
```

##### OUTPUT #####

```
Enter a string : this is a book.↵
koob a si siht
```

Ex. 8.

Program to reverse a string without reversing the word

```
#include<stdio.h>
main()
{
    char str[100];
    int i,temp;
    clrscr();
    printf ("Enter a string : ");
    gets(str);
    fflush(stdin);
    for(i=0;str[i];i++);
    for(i--;i>=-1;i--)
    {
        if(str[i]==' ' || str[i]=='\t' || i==-1)
        {
            temp=i;
            for(i++;str[i]!='\0' && str[i]!=' ' && str[i]!='\t';i++)
                printf ("%c",str[i]);
            i=temp;
            printf (" ");
        }
    }
}
```

## 'C' Scope

```
}  
getch();  
}
```

##### OUTPUT #####

Enter a string : this is a book.↵  
book a is this

Ex. 9.

Program to reverse a string without reversing the word

```
#include<stdio.h>  
main()  
{  
char str[100];  
int i,temp;  
clrscr();  
printf ("Enter a string : ");  
gets(str);  
fflush(stdin);  
for(i=0;str[i];i++);  
for(i--;i>=-1;i--)  
{  
if(str[i]==' '||str[i]=='\t' || i==-1)  
{  
temp=i;  
for(i++;str[i]!='\0'&&str[i]!=' ';&& str[i]!='\t' ;i++)  
printf ("%c",str[i]);  
i=temp;  
printf (" ");  
}  
}  
getch();
```

## 'C' Scope

```
}
```

```
##### OUTPUT #####
```

```
Enter a string : this is a book  
book a is this
```

Ex. 10.

Program to form Piglatin of a string general ( it is "pig latin like" version of this string by using the following rule: move the first character to the end of the word and append "a".)

```
#include<stdio.h>
```

```
main()  
{  
  char str[100];  
  int i;  
  char temp;  
  clrscr();  
  printf ("Enter a string : ");  
  gets(str);  
  fflush(stdin);  
  temp=str[0];  
  for(i=1;str[i];i++)  
  {  
    if(str[i]==' ' || str[i]=='\t' || str[i]=='\0')  
    {  
      printf ("%ca ",temp);  
      temp=str[++i];
```

## 'C' Scope

```
}  
else  
{  
    printf ("%c",str[i]);  
}  
}  
printf ("%ca ",temp);  
getch();  
}
```

##### OUTPUT #####

Run part of above program  
Enter a string : this is a book.↵  
hista sia aa ookba

Ex. 11.

Program to check whether the string is palindrome or not.  
(A string is said to be palindrome string if reversing the string gives the same string as result.)

```
#include<stdio.h>  
main()  
{  
    char str[100];  
    int i,j;  
    clrscr();  
    printf ("Enter a string : ");  
    gets(str);  
    fflush(stdin);  
    for(i=1;str[i];i++);  
    for (i--,j=0;i>=0 && str[i]==str[j];i--,j++);  
    if(i== -1)
```

## 'C' Scope

```
{
printf ("Entered String is a Palindrome : ");
}
else
{
printf ("Entered String is not a palindrome : ");
}
getch();
}
```

##### OUTPUT #####

```
Enter a string : this
Entered String is not a palindrome :
Enter a string : level
Entered String is a Palindrome :
```

Ex. 12.

Program to find frequency count of a substring into a parent string

```
#include<stdio.h>
main()
{
char str[100],fstr[20];
int i,j,count=0;
clrscr();
printf("Enter the string : ");
gets(str);
fflush(stdin);
printf ("Enter the string to count : ");
gets(fstr);
```

## 'C' Scope

```
fflush(stdin);
for(i=0;str[i];i++)
{
    if(str[i]==fstr[0])
    {
        for(j=0;str[i]==fstr[j] && fstr[j];i++,j++);
        if(!fstr[j])
        {
            count++;
        }
    }
}
printf ("The string [ %s ] occurs in [ %s ] %d times
",fstr,str,count);
getch();
}
```

##### OUTPUT #####

Run part of above program

Enter the string : this is a book.↵

Enter the string to count : is.↵

The string [ is ] occurs in [ this is a book ] 2 times

Ex. 13.

Program to count the frequency of a string in a long string  
advance

```
#include<stdio.h>
main()
{
    char str[100],fstr[20],ans;
```

## 'C' Scope

```
int i,j,count=0;
clrscr();
printf("Enter the string : ");
gets(str);
fflush(stdin);
printf ("Enter the string to count : ");
gets(fstr);
fflush(stdin);
printf ("Match Whole Word : ");
ans=getchar();
fflush(stdin);

for(i=0;str[i];i++)
{
if(ans=='y')
{
if(str[i]==fstr[0] && str[i-1]!='')
{
for(j=0;str[i]==fstr[j] && fstr[j];i++,j++);
if(!fstr[j] && str[i]=='')
{
count++;
}
}
}
else
{
if(str[i]==fstr[0])
{
for(j=0;str[i]==fstr[j] && fstr[j];i++,j++);
if(!fstr[j])
{
count++;
}
```

## 'C' Scope

```
    }  
    }  
    }  
    }  
    printf ("The string [ %s ] occurs in [ %s ] %d times  
",fstr,str,count);  
    getch();  
}
```

##### OUTPUT #####

Enter the string : this is a good book published by scope  
computer

Enter the string to count : is

Match Whole Word : y

The string [ is ] occurs in [ this is a good book published by  
scope computer ]

1 times

Enter the string : this is a book published by scope  
computer.↵

Enter the string to count : is.↵

Match Whole Word : n.↵

The string [ is ] occurs in [ this is a book published by  
scope computer ]

3 times

Ex. 14.

Program to find and replace the string

```
#include<stdio.h>  
main()  
{
```



## 'C' Scope

```
char str[100],fstr[20],rstr[20],ans;
int i,j,count=0;
clrscr();
printf("Enter the string : ");
gets(str);
fflush(stdin);
printf ("Enter the string to count : ");
gets(fstr);
fflush(stdin);
printf ("Enter the string to replace : ");
gets(rstr);
fflush(stdin);
printf ("Match Whole Word : ");
ans=getchar();
fflush(stdin);

for(i=0;str[i];i++)
{
    if(ans=='y')
    {
        if(str[i]==fstr[0] && str[i-1]!=' ')
        {
            for(j=0;str[i]==fstr[j] && fstr[j];i++,j++);
            if(!fstr[j] && str[i]!=' ')
            {
                printf ("%s",rstr);
            }
        }
        printf ("%c",str[i]);
    }
    else
    {
        if(str[i]==fstr[0])
```

## 'C' Scope

```
{
  for(j=0;str[i]==fstr[j] && fstr[j];i++,j++);
  if(!fstr[j])
  {
    printf ("%s",rstr);
  }
}
printf ("%c",str[i]);
}
}
getch();
}
```

##### OUTPUT #####

Enter the string : this is a good book published by scope  
computer.↵

Enter the string to count : is.↵

Enter the string to replace : are.↵

Match Whole Word : n.↵

there are a book publarehed by scope computer

Enter the string : this is a book published by scope  
computer.↵

Enter the string to count : is.↵

Enter the string to replace : are.↵

Match Whole Word : y.↵

this are a book published by scope computer

Ex. 15.

Program to find and replace the string (advance version)

## 'C' Scope

```
#include<stdio.h>
main()
{
    char ans1,str[100],fstr[20],rstr[20],tstr[100],ans;
    int i,j,t,count=0,c=1,r=1;
    clrscr();
    printf("Enter the string : ");
    gets(str);
    fflush(stdin);
    printf ("Enter the string to count : ");
    gets(fstr);
    fflush(stdin);
    printf ("Enter the string to replace : ");
    gets(rstr);
    fflush(stdin);
    printf ("Match Whole Word : ");
    ans=getchar();
    fflush(stdin);
    getch();
    clrscr();
    gotoxy(1,2);
    printf ("%s\n",str);
    getch();
    for(i=0,t=0;str[i];t++,i++)
    {
        if(ans=='y')
        {
            if(str[i]==fstr[0] && str[i-1]==' ')
            {
                for(j=0;str[i]==fstr[j] && fstr[j];i++,j++);
                if(!fstr[j] && str[i]==' ')
                {
```

## 'C' Scope

```
gotoxy(i,1);
textattr(142);
cprintf("%c",25);
gotoxy(55,1);
printf ("Replace Y/N : ");
ans1=getche();
fflush(stdin);
gotoxy(i,1);
printf (" ");
if(ans1=='y')
{
for(r=0;rstr[r];r++)
{
tstr[t++]=rstr[r];
}
}
else
{
for(r=0;fstr[r];r++)
{
tstr[t++]=fstr[r];
}
}
}
}
tstr[t]=str[i];
}
else
{
gotoxy(c,r);
if(str[i]==fstr[0])
{
for(j=0;str[i]==fstr[j] && fstr[j];i++,j++);
```

## 'C' Scope

```
if(!fstr[j])
{
    gotoxy(i,1);
    printf("%c",25);
    gotoxy(55,1);
    printf ("Replace Y/N : ");
    ans1=getche();
    fflush(stdin);
    if(ans1=='y')
    {
        for(r=0;rstr[r];r++)
        {
            tstr[t++]=rstr[r];
        }
    }
    else
    {
        for(r=0;fstr[r];r++)
        {
            tstr[t++]=fstr[r];
        }
    }
}
tstr[t]=str[i];
}
tstr[t]='\0';
gotoxy(3,3);
printf ("%s",tstr);
getch();
}
```

## 'C' Scope

##### OUTPUT #####

this is a book

This is a good book

Run part of above program:=

Enter the string : this is a good book published by scope  
computer

Enter the string to count : is

Enter the string to replace : are

Match Whole Word : n

↓ ↓

↓

Replace Y/N :

n

this is a good book published by scope computer

there is a good book published by scope computer

Enter the string : this is a good book published by scope  
computer

Enter the string to count : is

Enter the string to replace : are

Match Whole Word : y

↓

Replace Y/N : y

this is a good book published by scope computer

this are a good book published by scope computer

Ex. 16.

Program to print a string as triangle, like one shown below

S

SC

SCO

SCOP

## 'C' Scope

SCOPE

```
#include<stdio.h>
main()
{
char str[50]={"SCOPECOMPUTER"};
int i,len;
clrscr();
for (i=1;i<=strlen(str);i++)
printf ("\n%-.*s",i,str);
getch();
}
```

##### OUTPUT #####

```
S
SC
SCO
SCOP
SCOPE
SCOPEC
SCOPECO
SCOPECOM
SCOPECOMP
SCOPECOMPU
SCOPECOMPUT
SCOPECOMPUTE
SCOPECOMPUTER
```

Ex. 17.

Program to print a string as reverse triangle as shown below

## 'C' Scope

SCOPE  
SCOP  
SCO  
SC  
S

```
#include<stdio.h>
main()
{
char str[50]="SCOPECOMPUTER";
int i,len=strlen(str);
clrscr();
for (i=1;i<=len;i++)
printf ("\n%-*.*s",len,len-i+1,str);
getch();
}
```

##### OUTPUT #####

SCOPECOMPUTER  
SCOPECOMPUTE  
SCOPECOMPUT  
SCOPECOMPU  
SCOPECOMP  
SCOPECOM  
SCOPECO  
SCOPEC  
SCOPE  
SCOP  
SCO  
SC  
S



## 'C' Scope

Ex. 18.

Program to print a string as reverse triangle as shown below

```
SCOPE
 SCOP
  SCO
   SC
    S
```

```
#include<stdio.h>
main()
{
char str[50]="SCOPECOMPUTER";
int i,len=strlen(str);
clrscr();
for (i=1;i<=len;i++)
printf ("\n%*.*s",len,len-i+1,str);
getch();
}
```

##### OUTPUT #####

```
SCOPECOMPUTER
 SCOPECOMPUTE
  SCOPECOMPUT
   SCOPECOMPU
    SCOPECOMP
     SCOPECOM
      SCOPECO
       SCOPEC
        SCOPE
```

## 'C' Scope

```
SCOP
SCO
SC
S
```

Ex. 19.

Program to print scope computer as a reverse triangle like below

```
S
SC
SCO
SCOP
SCOPE
```

```
#include<stdio.h>
main()
{
char str[50]={"SCOPECOMPUTER"};
int i,len=strlen(str);
clrscr();
for (i=1;i<=len;i++)
printf ("\n%*. *s",len,i,str);
getch();
}
```

##### OUTPUT #####

```
S
SC
SCO
SCOP
SCOPE
```

## 'C' Scope

SCOPEC  
SCOPECO  
SCOPECOM  
SCOPECOMP  
SCOPECOMPU  
SCOPECOMPUT  
SCOPECOMPUTE  
SCOPECOMPUTER

### 2-D INTEGER ARRAY

Friends previously we have read about one dimensional integer array which is used for list of numbers. Now consider the situation where you have 3 patients, and all the patients are monitored for the body temperature for 4 days. The temperature of all the patients for four days is given in following table.

	0	1	2	3	
	Day 1	Day 2	Day 3	Day 4	
0	Patient no.1	93	95	97	98
1	Patient no.2	100	102	103	99
2	Patient no 3	98	101	97	96

This table contains 12 values of body temperature with 4 values in each row. Now consider this table as row and

## 'C' Scope

columns collection, there are 3 rows and 4 columns, which is now represented by a 3 X 4 matrix.

Now, if we want to extract any one value of temperature we have to mention the row no. and column no. Suppose that the entire matrix is denoted by variable m and if you want to extract the temperature of patient no. 2 of day 3. This temperature can be extracted by `m[1][2]`. This `m[1][2]` refers to temperature 103.

To define m we can use the following syntax

```
int m[3][4];
```

This is what we can represent by a two dimensional integer array.

We can represent these values in terms of memory layout as shown below

0				1				2			
93	95	97	98	100	102	103	99	98	101	97	96
0	1	2	3	0	1	2	3	0	1	2	3
First row of array				Second row of array				Third row of array			

According to the given conditions, first value will be stored at `m[0][0]` second will be stored at `[0][1]` third will be at `m[0][2]`, fourth value will be stored at `m[0][3]`, these values are for patient one. In the same style value will be stored for patient no.2 i.e temperature of day 1 for patient 2

## 'C' Scope

will be stored at `m[1][0]` for second day it will be stored at `m[1][1]` and so on.

We can also initialize these values as given below

```
int m[3][4]={93,95,97,98,
             100,102,103,99,
             98,101,97,96};
```

We can also input the values at run time

Ex. 20.

Program to implement use of 2D array.

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
int i,j,m[3][4];
```

```
clrscr();
```

```
for(i=0;i<3;i++)
```

```
{
```

```
for(j=0;j<4;j++)
```

```
{
```

```
printf ("Enter the value for patient no. %d and day no. %d",i+1,j+1);
```

```
scanf ("%d",&m[i][j]);
```

```
}
```

```
}
```

```
printf ("\n\nTemperatures of the patients \n\n");
```

```
printf ("\n
```

```
day 1 day 2 day 3 day
```

```
4\n\n");
```

```
for(i=0;i<3;i++)
```

```
{
```

## 'C' Scope

```
printf("Temperature of patient no. %d",i+1);
for(j=0;j<4;j++)
{
    printf("    %d    ",m[i][j]);
}

printf("\n");
}
getch();
}
```

##### OUTPUT #####

Enter the value for patient no. 1 and day no. 1 93  
Enter the value for patient no. 1 and day no. 2 95  
Enter the value for patient no. 1 and day no. 3 97  
Enter the value for patient no. 1 and day no. 4 98  
Enter the value for patient no. 2 and day no. 1 100  
Enter the value for patient no. 2 and day no. 2 102  
Enter the value for patient no. 2 and day no. 3 103  
Enter the value for patient no. 2 and day no. 4 99  
Enter the value for patient no. 3 and day no. 1 98  
Enter the value for patient no. 3 and day no. 2 101  
Enter the value for patient no. 3 and day no. 3 97  
Enter the value for patient no. 3 and day no. 4 96

Temperatures of the patients

	day 1	day 2	day 3	day 4
Temperature of patient no. 1	93	95	97	98
Temperature of patient no. 2	100	102	103	99

## 'C' Scope

Temperature of patient no. 3 98 101 97 96

Ex. 21.

Program to convert rows to columns and columns to rows in a matrix

```
#include<stdio.h>
main()
{
char str[4][4];
int a,b,total=0;
clrscr();
for(a=0;a<=3;a++)
{
for(b=0;b<=3;b++)
{
printf("Enter the element of %d row and %d column:=",
",a+1,b+1);
scanf("%d",&str[a][b]);
}
printf("\n");
}

for(a=0;a<=3;a++)
{
for(b=0;b<=3;b++)
printf(" %d",str[a][b]);
printf("\n");
}
printf("\n");
for(b=0;b<=3;b++)
{
for(a=0;a<=3;a++)
```

## 'C' Scope

```
printf(" %d",str[a][b]);  
printf("\n");  
}
```

```
getch();  
}
```

##### OUTPUT #####

Enter the element of 1 row and 1 column:= 2  
Enter the element of 1 row and 2 column:= 5  
Enter the element of 1 row and 3 column:= 3  
Enter the element of 1 row and 4 column:= 7

Enter the element of 2 row and 1 column:= 5  
Enter the element of 2 row and 2 column:= 9  
Enter the element of 2 row and 3 column:= 1  
Enter the element of 2 row and 4 column:= 8

Enter the element of 3 row and 1 column:= 4  
Enter the element of 3 row and 2 column:= 2  
Enter the element of 3 row and 3 column:= 7  
Enter the element of 3 row and 4 column:= 1

Enter the element of 4 row and 1 column:= 9  
Enter the element of 4 row and 2 column:= 4  
Enter the element of 4 row and 3 column:= 8  
Enter the element of 4 row and 4 column:= 3

```
2 5 3 7  
5 9 1 8  
4 2 7 1  
9 4 8 3
```



## 'C' Scope

```
2 5 4 9
5 9 2 4
3 1 7 8
7 8 1 3
```

Ex. 22.

Program to insert elements in matrix and print the row total, column total and diagonal total

```
#include<stdio.h>
main()
{
    char str[4][4];
    int a,b,total=0;
    clrscr();
    for(a=0;a<=3;a++)
    {
        for(b=0;b<=3;b++)
        {
            printf("Enter the element of %d row and %d column:=",
"a+1,b+1);
            scanf("%d",&str[a][b]);
        }
        printf("\n");
    }
    for(a=0;a<=3;a++)
    {
        total =0;
        for(b=0;b<=3;b++)
        {
            total+=str[a][b];
```

## 'C' Scope

```
printf(" %d",str[a][b]);  
}  
printf(" %d\n",total);  
}
```

```
for(b=0;b<=3;b++)  
{  
total =0;  
for(a=0;a<=3;a++)  
{  
total+=str[a][b];  
}  
printf(" %d",total);  
}
```

```
total = 0;  
for(a=0;a<=3;a++)  
{  
total+=str[a][a];  
}  
printf(" %d",total);  
gotoxy(13,20);  
total = 0;  
for(a=0,b=3;a<=3;a++,b--)  
{  
total+=str[a][b];  
}  
printf(" %d",total);  
getch();  
}
```

#####OUTPUT#####

## 'C' Scope

Run part of above program

Enter the element of 1 row and 1 column:= 2

Enter the element of 1 row and 2 column:= 5

Enter the element of 1 row and 3 column:= 1

Enter the element of 1 row and 4 column:= 8

Enter the element of 2 row and 1 column:= 2

Enter the element of 2 row and 2 column:= 3

Enter the element of 2 row and 3 column:= 9

Enter the element of 2 row and 4 column:= 4

Enter the element of 3 row and 1 column:= 5

Enter the element of 3 row and 2 column:= 8

Enter the element of 3 row and 3 column:= 3

Enter the element of 3 row and 4 column:= 7

Enter the element of 4 row and 1 column:= 2

Enter the element of 4 row and 2 column:= 8

Enter the element of 4 row and 3 column:= 4

Enter the element of 4 row and 4 column:= 7

2 5 1 8 16

2 3 9 4 18

5 8 3 7 23

2 8 4 7 21

11 24 17 26 15

## **2-D Character Array**

Now that we have read enough about 2D integer array and one dimensional character array now we will learn two dimensional character array.

## 'C' Scope

Consider the following

```
char nm[10];
```

This statement can be used to store a single name in variable nm, but consider the situation that you want to store more than one names refers to single variable then you have to use two dimensional character array.

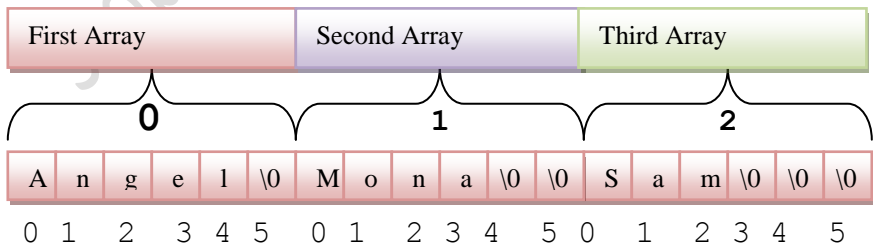
Suppose you want to store 3 names referred by a variable nm, then we will declare the variable as given below.

```
char nm[3][6];
```

This statement will occupy 3 different array of size 6 each.

We can also initialize this 2 Dimensional array as mentioned here

```
char nm[3][6]={
    "Angel",
    "Mona",
    "Sam"
};
```



## 'C' Scope

Unused memory blocks will be initialized by NULL. i.e nm[0][5], nm[1][4], nm[1][5], nm[2][3], nm[2][4] and nm[2][5] will be initialized by NULL character.

```
#include<stdio.h>
main()
{
    char arr[10][10]= { "Zero\0",
                       "One\0",
                       "Two\0",
                       "Three\0",
                       "Four\0" ,
                       "Five\0",
                       "Six\0",
                       "Seven\0",
                       "Eight\0",
                       "Nine\0"
                     };
    int num,temp,div =1,value;
    clrscr();
    printf("Enter the number:=");
    scanf("%d",&num);
    temp=num;
    for(;temp>=1;temp=temp/10)
        div*=10;
    div/=10;
    printf("divisor = %d\n",div);
    if(num==0)
        printf("zero");
    while (div>=1)
    {
        value =num/div;
        num%=div;
```

## 'C' Scope

```
div/=10;
printf(" %s ",arr[value]);
}
getch();
}
```

##### OUTPUT #####

```
Enter the number:=8904
divisor = 1000
Eight Nine Zero Four
```

Ex. 23.

Program to accept a three digit number and print it in words  
(like 123 => One hundred Twenty Three)

```
#include<stdio.h>
main()
{
    char arr[20][10]=
        {"zero\0","one\0","two\0","three\0","four\0",
         "five\0","six\0","seven\0","ei
         ght\0","nine\0",
         "ten\0","eleven\0","twelve\0"
         ,"thirteen\0",
         "forteen\0","fifteen\0","sixtee
         n\0",
         "seventeen\0","eighteen\0","n
         ineteen\0"};
    char sty[8][10]={"twenty\0","thirty\0","forty\0","fifty\0",
                    "sixty\0","seventy\0",
                    "eighty\0","ninety\0"};
```

## 'C' Scope

```
int num,temp,div =1,value;
clrscr();
printf("Enter the number:=");
scanf("%d",&num);
fflush(stdin);
temp=num;
for(temp/=10;temp>=1;temp/=10)
    div*=10;
printf("divisor = %d\n",div);
if(num==0)
    printf("zero");
while (num>=1)
{
    value =num/div;
    if(div==100)
        printf(" %s hundred ",arr[value]);
    else
    {
        if((num >=20 && num <=99) )
            printf(" %s ",sty[value-2]);
        else
        {
            printf("%s ",arr[num]);
            break;
        }
    }
    num%=div;
    div/=10;
}
getch();
}
```

## 'C' Scope

---

##### OUTPUT #####

Enter the number:=100

divisor = 100

one hundred

Enter the number:=120

divisor = 100

one hundred twenty

Enter the number:=113

divisor = 100

one hundred thirteen

Enter the number:=0

divisor = 1

zero

Enter the number:=10

divisor = 10

ten

Enter the number:=196

divisor = 100

one hundred ninety six

### Summary

- An Array is a collection of homogenous sequential memory locations named using single variable and differentiated using different indices.
- During the declaration of array we have to give the size i.e the total memory location to be occupied within square brackets([ ]).
- The size given in square brackets is called subscript. Also during manipulation, storage or retrieval of array an index counter variable is taken to access the elements in an array.



## 'C' Scope

### Self Review

- Q1. Write a program to find largest, second largest , and third largest element of an array?
- Q2. Write a program to mean, mode, median of arrays?
- Q3. Write a program to find sum of products, products of sum of two arrays?
- Q4. Write a program to remove duplicates in an array?
- Q5. Write a program to number of vowels and consonants in a string?
- Q6. Write a program to implement sparse matrix?
- Q7. Write a program to check whether the matrix is symmetric matrix or not ?
- Q8. Write a program to find sum of diagonal elements of a square matrix?
- Q9. Write a program to print transpose of matrix?
- Q10. Write a program to find inverse of 3X3 matrix?
- Q11. Write a program to implement Travelling Salesman Problem using matrix?
- Q12. Write a program to print square matrix helically?

# **CHAPTER - 6**

## **POINTERS**

Friends, What is a pointer ?

A pointer can be defined as a variable which can contain the address of another variable.

A normal variable can not hold the address of another variable.

Consider the segment

```
int x,y;  
y=&x;
```

The above initialisation is incorrect because x and y are the normal integer variables, and we are trying to assign the address of x in variable y. Which is not possible, because a normal variable can not hold the address of another variable.

Pointers are the special variables have the capability to hold the address of other variables.

As other variables a pointer must be defined before it can be used.

```
int x,*y;
```

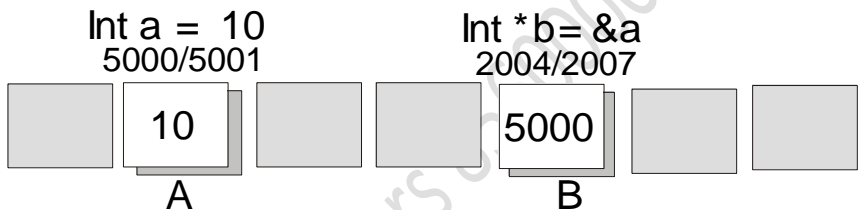
## 'C' Scope

```
y=&x;
```

In above segment we have declared y as pointer variable using asterisk sign, now y is a pointer variable, and it is containing the address of x.

(To declare a variable as a pointer variable simply place the asterisk in front the variable name).

The memory map of above segment can be



## The rvalues and lvalues

As we declare any variable, it stores into the memory. In 'C' we can say

Lvalue of a variable is the address of the variable.

And rvalue is the value that the variable hold

In the above example rvalue of b is equivalent to the lvalue of a as shown in memory map.

In above example the address of a is 5000 and the address of b is 2004 the value of a is 10 and the value of b is address of a i.e. 5000.

## 'C' Scope

---

Now let us see the output of the following program

```
main()
{
  int x=10;
  printf ("\n The value of x is : "x);
  printf ("\n The address of x is : " ,&x);
}
```

the output of the program would be

The value of x is : 10  
The address of the x is : 64450

## Void pointers :

Pointers can also be declared as void.

Void pointers can't be dereferenced without explicit casting. This is because the compiler can't determine the size of the object the pointer points to.

```
int x;
float r;
void *p = &x;      /* p points to x */
int main (void)
{
  *(int *) p = 2;
  p = &r;          /* p points to r */
  *(float *)p = 1.1;
}
```

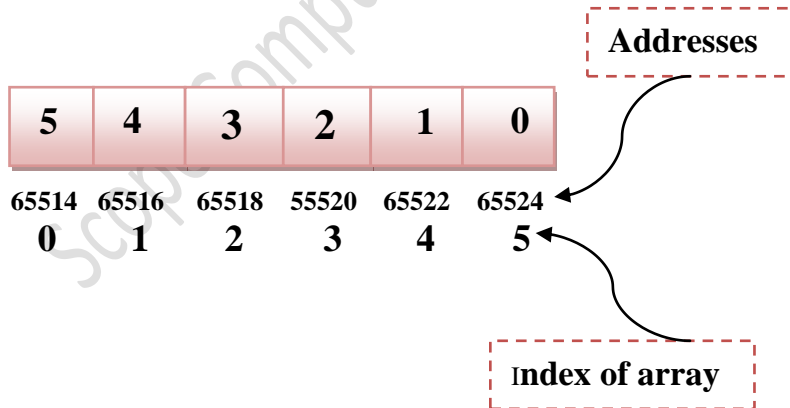
## 'C' Scope

```
#include <stdio.h>
main()
{
  int num[]={5,4,3,2,1,0},*pt;
  for(pt=&num[5];pt>=&num[0];pt--)
    printf ("\n%u          %d",pt,*pt);
  getch();
}
```

**\*\*output\*\***

```
65524      0
65522          1
65520          2
65518          3
65516          4
65514      5
```

Memory map for given program



**num**

## 'C' Scope

pt

65524

How to Understand ?

According to the given memory map and program, num is an array of 6 integers and pt is a pointer having the address of sixth element of the array which is 65524 for given condition. This address will not be the same for every execution of the program because the data may take different address at different times.

When the loop starts to execute for the first time the address of sixth element will be assigned to the pointer pt means the value for the pt will be 65524. First time it will print the value of pt i.e 65524 and the \*pt will print the value at 65524 i.e 0 in second chance value of pt will be reduced due to pt- - and it becomes 65522 and the output for the second time will be 65522 and \*pt will be 1. After executing printf() statement the value of pt will be reduced again and it will become 65520 and the output for the third time will be 65520 and 2. This process will be continued for six times. And the result will be printed as given below.

65524	0
65522	1
65520	2
65518	3
65516	4
65514	5

```
#include <stdio.h>
main()
```

## 'C' Scope

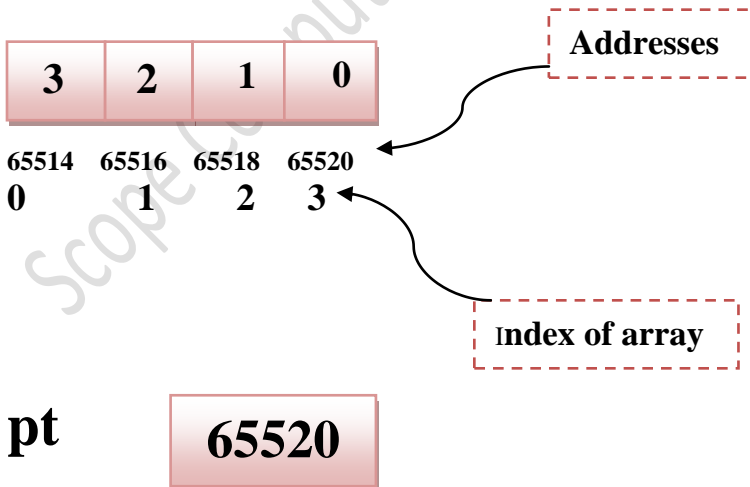
```
{
int a,num[]={3,2,1,0},*pt;
clrscr();
for(pt=num+3,a=*pt;a<=3;a++)
printf ("\n%d",pt[-a]);
getch();
}
```

\*\*\*output\*\*\*

0  
1  
2  
3

How to Understand ?

Memory map for given program



## 'C' Scope

**a**

**0**

How to Understand ?

According to the Given memory map first the array num will be initialized to the values 3,2,1,0. After the program control will go to the for loop in first iteration num+3 indicates the address of index 3 which is 65520 in this condition, so the 65520 will be assigned to pt.  $a=*pt$  will assign the value at address 65520 i.e. 0 to variable a, then control will enter to the condition part check the condition which is true and the  $pt[-a]$  will be printed

Meaning of  $pt[-a]$

$a=0$

$pt[-0] \rightarrow$  means  $*(pt-0) \rightarrow *(pt) \rightarrow 0$

second time a will become 1 and  $pt[-1]$  will be evaluated as  
 $pt[-1] \rightarrow$  means  $*(pt-1) \rightarrow *(65520-1) \rightarrow *(65518) \rightarrow 1$

third time value of variable a will become 2 and  $pt[-2]$  will be evaluated as

$pt[-2] \rightarrow$  means  $*(pt-2) \rightarrow *(65520-2) \rightarrow *(65516) \rightarrow 2$

four and final time value variable a will become 3 and  $pt[-3]$  will be evaluated as

$pt[-3] \rightarrow$  means  $*(pt-3) \rightarrow *(65520-3) \rightarrow *(65514) \rightarrow 3$

and finally the output will be

0



## 'C' Scope

1  
2  
3  
4

Ex. 1.

Program to implement pointer arithmetics

```
#include <stdio.h>
main()
{
    static int
    num[]={4,3,2,1,0}, *pt[]={num,num+1,num+2,num+3,num
    +4};
    int **p=pt;
    clrscr();
    printf ("\n%u",num);
    printf ("\n%d",*num);
    printf ("\n%u",pt);
    printf ("\n%u",*pt);
    printf ("\n%d",**pt);
    printf ("\n%u",p);
    printf ("\n%u",*p);
    printf ("\n%u",**p);
    p++;
    printf ("\n\n%d",p-pt);
    printf ("\n%d",*p-num);
    printf ("\n%d",**p);
    printf ("\n%d",*p[-0]);
    *p++;
    printf ("\n\n%d",p-pt);
    printf ("\n%d",*p-num);
    printf ("\n%d",**p);
```

## 'C' Scope

```
printf ("\n%d",*p[-1]);
*++p;
printf ("\n\n%d",p-pt);
printf ("\n%d",*p-num);
printf ("\n%d",**p);
printf ("\n%d",*p[-2]);
++*p;
printf ("\n\n%d",p-pt);
printf ("\n%d",*p-num);
printf ("\n%d",**p);
printf ("\n%d",*p[-3]);
getch();
}
```

\*\*\*\*output\*\*\*\*

170

4

180

170

4

180

170

4

1

1

3

3

2

2

## 'C' Scope

2

3

3

3

1

3

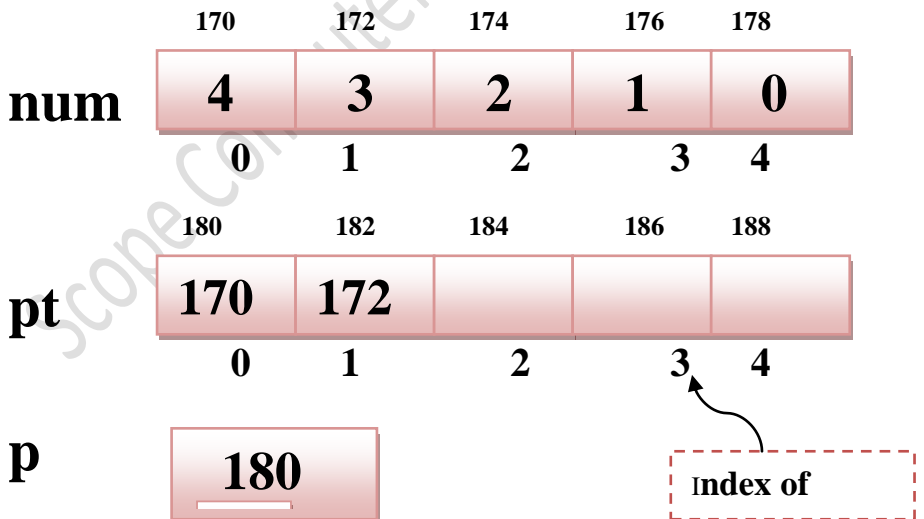
3

4

0

4

Memory map for given program



How to Understand ?

Start evaluating from line No. 7

## 'C' Scope

```
printf ("\n%u",num);
```

The value of num is the starting address of num i.e. 170 hence the output for this statement will be 170.

Now let us evaluate the line No. 8

```
printf ("\n%d",*num);
```

here the meaning of \* is value at address if we expand the statement \*num will become \*170 means the value at address 170 which is 4 the output of the line no. 8 will be 4

evaluate line no. 9

```
printf ("\n%u",pt);
```

pt will print the starting address of pt which is 180, therefore the output for the line no. 9 will be 180.

Evaluate line no. 10

```
printf ("\n%u",*pt);
```

here the meaning of \* is value at address if the we expand the statement \*pt will become \*180, therefore the value at address 180 which is 170 the output of the line no. 10 will be 170.

Evaluate line no. 11

```
printf ("\n%d",**pt);
```

In this statement pt has the starting address of array pt[] which is 180 \*\*pt can be expanded as \*(\*pt) here \*pt is

## 'C' Scope

170 and \*(170) (value at address 170) is 4 therefore the output of line no. 11 will be 4.

Evaluate line no. 12

```
printf ("\n%u",p);
```

Here p has the base address of array pt[] which is 180 there the output for the line no. 12 will be 180.

Evaluate line no. 13

```
printf ("\n%u",*p);
```

Here p has the value 180 the \*p means value at 180 which is 170 therefore the output of line no. 13 will be 170

Evaluate line no. 14

```
printf ("\n%u",**p);
```

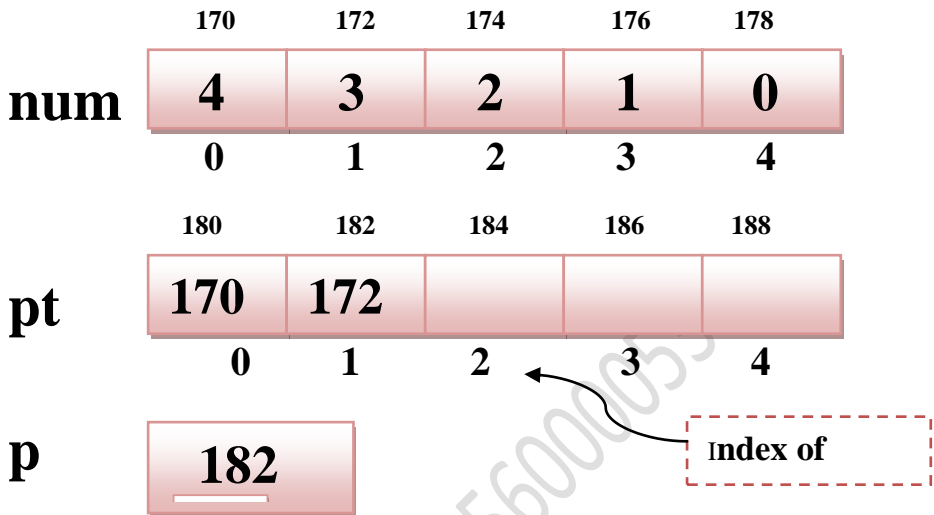
in line no. 13 we have evaluated \*p which is 170 and \*\*p can be expanded as \*(170) which is 4 hence the output of line no. 14 will be 4.

Evaluate the line no. 15

```
p++;
```

this will increase the value of p and the value of p will become 182 and the memory map will be change like this

## 'C' Scope



now evaluate the line no. 16

```
printf ("\n\n%d",p-pt);
```

Now try to evaluate the meaning of  $p-pt$ . in this statement  $p$  is containing the value 182. And  $pt$  is containing the value base address of  $pt$  which is 180, and hence  $182-180$  will be evaluated and the output of line no. 16 will be 1. ( because 1 integer occupies two bytes of memory).

Now evaluate the line no. 17.

```
printf ("\n%d",*p-num);
```

in line no. 17  $p$  contains the value 182 and  $*(182)$  is 172. Num contains the base address of array  $num[]$  which is 170 hence the output of  $*p-num$  will be  $172-170$  which yields 1.

## 'C' Scope

Now evaluate line no. 18

```
printf ("\n%d",**p);
```

in previous statement we have checked the \*p is 172 and value at 172 is 3 the expression can be expanded as  $*(*(182))$  therefore the output will be 3.

Now evaluate line no. 19

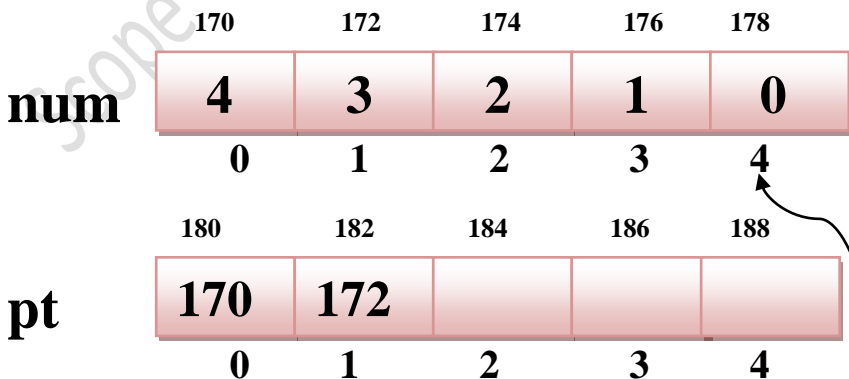
```
printf ("\n%d",*p[-0]);
```

now expand  $p[-0]$  as  $*(p-0)$  which is \*p and the  $*p[-0]$  will  $**p$ . and value at 182 is 172 and the value at 172 is 3 and therefore the output of line 19 will be 3.

Evaluate line no. 20

```
*p++;
```

This will increase the value of p from 182 to 184 and memory map will be changed like this



## 'C' Scope

**p**

**184**

index of



now evaluate line no. 21

```
printf ("\n\n%d",p-pt);
```

in this statement p has the value 184 and pt contains the base address of array pt[] which is 180. After evaluating 184-180 the output will be 2.

Now evaluate line no. 22

```
printf ("\n%d",*p-num);
```

Meaning of \*p is the value at address 184 which is 174 and meaning of num is base address of array num[] which is 170. Now \*p-num will be 174 - 170 which yields 2.

Now evaluate line no. 23

```
printf ("\n%d",**p);
```

in previous statement we have checked the \*p is 174 and value at 174 is 2 the expression can be expanded as \*(\*(184)) therefore the output will be 2.

Now evaluate line no. 24

```
printf ("\n%d",*p[-1]);
```

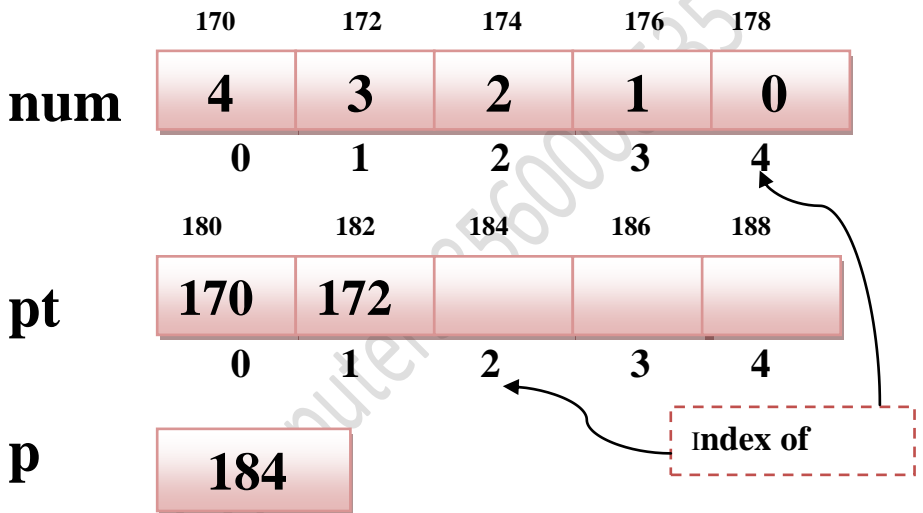


## 'C' Scope

now expand  $p[-1]$  as  $*(p-1)$  which is  $*(184-1)$  which is  $*(182)$  and the  $*p[-1]$  will  $** (182)$ . Which is 3.

Now evaluate line no. 25

`*++p;`



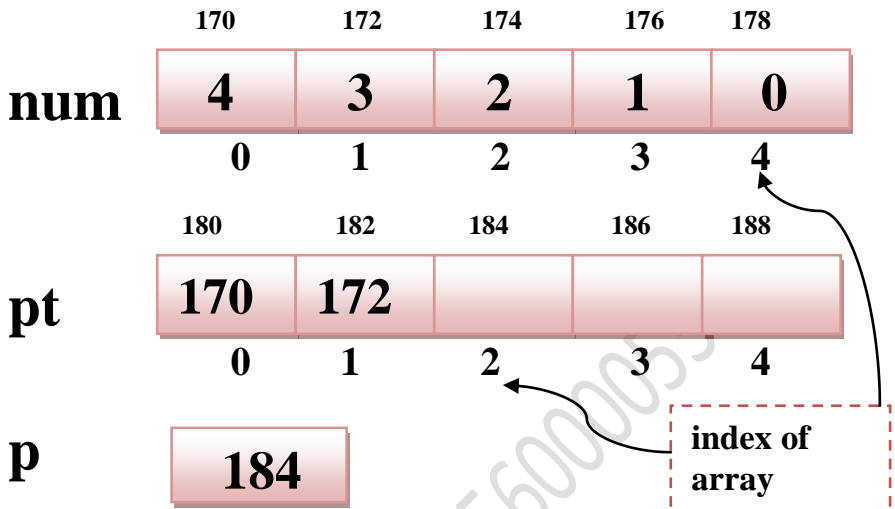
This ++ increase the value of p from 184 to 186. After this we are using \* which is not used for any other operation so this \* has to be ignored.

Now that you have practiced so many pointer expressions, you can easily evaluate the result of statement nos. 26,27,28 and 29 respectively.

Now evaluate statement no. 30

`++*p;`

## 'C' Scope



In this statement preference will be given to \*. So this expression will increment the value given by \*p. and the \*p is 176 which will be incremented and 176 will become 178 as shown in memory map.

Now try to evaluate the statements no. 31, 32, 33 and 34.

```
#include <stdio.h>
main()
{
int num[3][3], *pt, i, j, k=0;
clrscr();
pt=num;

for(i=0; i<3; i++)
{
for(j=0; j<3; j++)
{
```

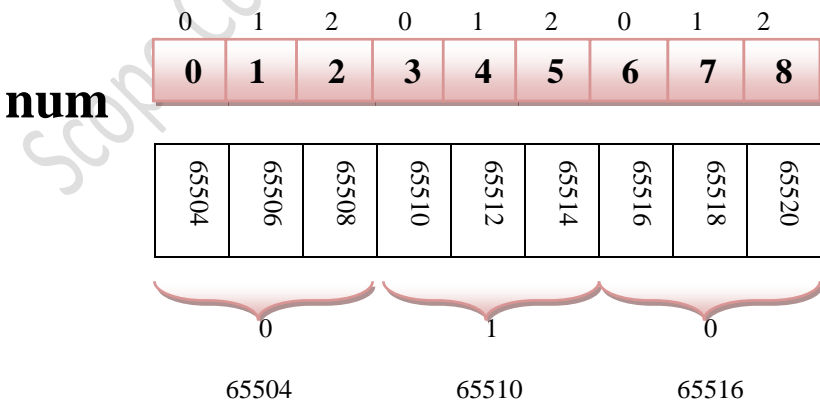
## 'C' Scope

---

```
    num[i][j]=k++;  
    }  
    }  
    printf ("\n%u",num);  
    *pt++;  
    printf ("\n%u",pt);  
    printf ("\n%u",num+1);  
    printf ("\n%d",pt[1]);  
    printf ("\n%u",num[1]);  
    printf ("\n%d",*(pt+1));  
    getch();  
    }
```

\*\*\*\*output\*\*\*\*

65504  
65506  
65510  
2  
65510  
2



## 'C' Scope

**pt**

**65504**

Statement no. 14

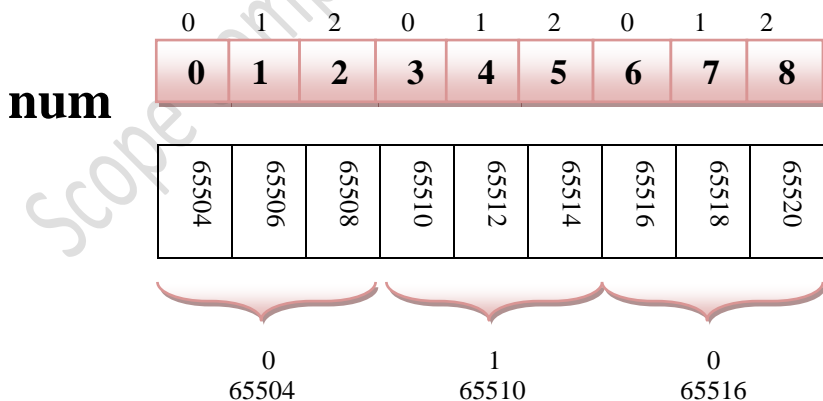
```
printf ("\n%u",num);
```

num is an array so only num will contain the base address of the array, and hence the output would be 65504 which is the base address of num.

statement no. 15

```
*pt++
```

Value stored at pt will be incremented this will become 65506. And asterisk will be ignored because after increment we are not using this for any other operation.



**pt**

**65504**

## 'C' Scope

statement no. 16

```
printf ("\n%u",pt);
```

this will print the value of pt i.e 65506.

Statement no. 17

```
printf ("\n%u",num+1);
```

since variable contains the base address of num i.e 65504 and this is the 2nd array and if we increase the value by one the this will go to the address of second array i.e. 65510.

Statement no. 18

```
printf ("\n%d",pt[1]);
```

now expand pt[1] i.e.  $*(pt+1) \Rightarrow *(65506+1) \Rightarrow *(65508)$   
ie. 2

statement no. 19

```
printf ("\n%u",num[1]);
```

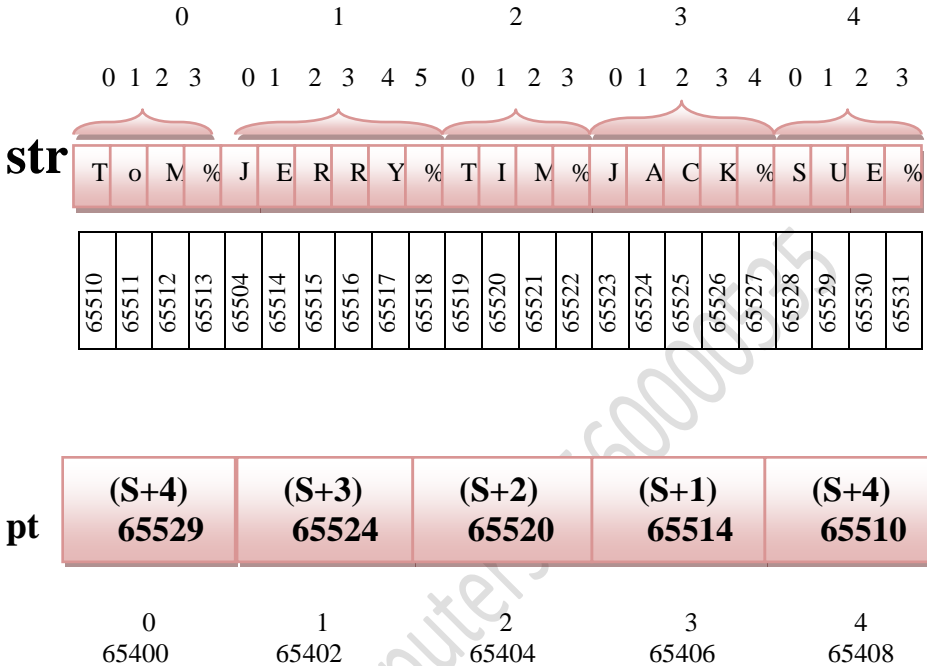
this will print the base address of num[1] ie. 65510.

Statement no. 20

```
printf ("\n%d",*(pt+1));
```

This will print the value at (65506+1) i.e value at 65508 ie.  
2.

## 'C' Scope



**S**

**65400**

```
#include <stdio.h>
main()
{
    static char *str[]={"tom","jerry","tim","jack","sue"};
    static char **pt[]={str+4,str+3,str+2,str+1,str};
    char ***s=pt;
    clrscr();
    printf ("\n line no.  %s",**s++ + 2 );
    printf ("\nline no. 1 %s",**++s);
    printf ("\nline no. 2 %s",*--*++s + 2);
}
```

## 'C' Scope

```
printf ("\nline no. 3 %s",*s[-2]+2);  
printf ("\nline no. 4 %s",s[-2][-1]+2);  
getch();  
}
```

line no. e

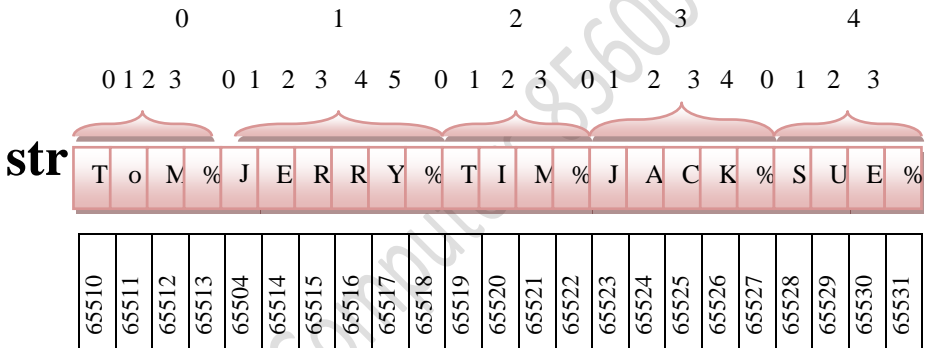
line no. 1 tim

line no. 2 m

line no. 3 ck

line no. 4 m

Now we will see how this memory map will be made. First we will see the memory map of STR



In the declaration part of the program STR will work as 2D array and the string S will occupy the memory according to its contents which is shown in figure

TOM will occupy 4 bytes of memory.

JERRY will occupy 6 bytes of memory.

TIM will occupy 4 bytes of memory.

JACK will occupy 5 bytes of memory.

SUE will occupy 4 bytes of memory.

## 'C' Scope

According to this figure the value of STR will be 65510 and STR +1 will be the base address of second dimension of STR which is 65514. Accordingly STR +2 will be the base address of third dimension of STR which is 65520, STR +3 will be the base address of fourth dimension of STR which is 65524, STR +4 will contain the base address of fifth dimension of STR which is 65529.

Now according to the second declaration statement i.e. static char \*\*pt[]={str+4,str+3,str+2,str+1,str};

The memory map will be as follows :-

<b>pt</b>	<b>(S+4)</b> <b>65529</b>	<b>(S+3)</b> <b>65524</b>	<b>(S+2)</b> <b>65520</b>	<b>(S+1)</b> <b>65514</b>	<b>(S+4)</b> <b>65510</b>
	0	1	2	3	4
	65400	65402	65404	65406	65408

And according to the third declaration statement i.e. char \*\*\*s=pt;

The memory map will be as follows :-

<b>S</b>	<b>65400</b>
	65300

statement no. 8

```
printf ("\n line no. %s",**s++ + 2 );
```

As per the statement value of s is 65400 so \*\*s will be 65529 and adding 2 to this address will extract the value at 65531 which yields the character 'e'. After printing the value 'e' ++ will be calculated because it is post increment operator.

Current value of S is now :-



## 'C' Scope

**S**

**65402**

65300

statement no. 9

```
printf ("\nline no. 1 %s",**++s);
```

now split this statement as ++s will increment the value to 65404 and value at this address is 65520 and further the next \* will yield the value at this address 65520 i.e. string 'TIM'

Current value of S is now :-

**S**

**65404**

65300

statement no. 10

```
printf ("\nline no. 2 %s",*--*++s + 2);
```

Splitting the above statement first ++s will be calculated and the value of s will become 65406. Now value at this address is 65514. After calculating – will send the control to the previous dimension which is 65510 and +2 will print value at address 65512 i.e. the character 'M' .

Current value of S is now :-

**S**

**65406**

65300

statement no. 11

## 'C' Scope

```
printf ("\nline no. 3 %s",*s[-2]+2);
```

Expanding the above statement the  $*s[-2]$  we get  $*(s-2)$  and  $*(s-2)$  will be calculated first and the value of  $*(s)$  will become 65402. Now the value at 65402 is 65524 and after adding 2 control will be transferred to 65526 and will print the value at address i.e. string "CK"

Current value of S is now :-

**S**

**65406**

65300

statement no. 12

```
printf ("\nline no. 4 %s",s[-2][-1]+2);
```

Expanding the above statement we get  $*(s-2)[-1] + 2$ . Now subtracting 2 the value of s becomes 65402. The value at this address is 65524. Now subtracting 1 we get the address of previous dimension i.e. 65520. Now finally adding 2 to this address the value at 65522 will print 'm'.

## Summary

- A pointer can be defined as a variable which can contain the address of another variable.
- A normal variable can not hold the address of another variable.
- Pointers are the special variables have the capability to hold the address of other variables.
- Void pointers can't be dereferenced without explicit casting. This is because the compiler can't determine the size of the object the pointer points to.

## 'C' Scope

---

### Self Review

- Q1. What is the difference between null pointers and dangling pointers?
- Q2. What is the difference between near & far pointer?
- Q3. Explain the meaning of pointer to pointer?
- Q4. What your opinion about pointers? Are they boon or bane? Justify your answer?
- Q5. Write a program to sort an array using pointers?
- Q6. Write a program to input & manipulate multidimensional array using pointers? Preferably perform matrix multiplication using pointers?
- Q7. Write a program to print Fibonacci series using pointers?

# CHAPTER – 7

## Functions

Students..! You always used main() in every 'C' Program.

What is main() ?

The answer is that it is a function. In 'C' Every Program code is written in a boundary called function.

What is a function?

It is a Self contained Block of Program Which performs a Specific Task.

In 'C' There are two types of functions

[1] Library Defined

[2] User Defined

### **[1] Library Defined Functions**

Library defined Functions are provided by the 'C' compiler and user can use these functions in their program wherever they want any number of times freely and for every function the appropriate header file must be included.

e.g pow() is a library function which is declared in math.h header file and returns the power.

## 'C' Scope

```
x=pow(2,3);  
printf ("%d",x);
```

will print 8 i.e. ( $2^3$ )

e.g sqrt() is a library function declared in math.h library file that returns square root of a number.

Ex. 1.

Program to find solution of a Quadratic Equation

```
#include <math.h>  
main()  
{  
float a,b,c,d,r1,r2;  
clrscr();  
printf ("Enter the value for a :");  
scanf ("%f",&a);  
printf ("Enter the value for b :");  
scanf ("%f",&b);  
printf ("Enter the value for c :");  
scanf ("%f",&c);  
d=pow(b,2)-4 * a *c;  
if (d< 0)  
printf ("Roots are imaginary : " );  
else  
{  
r1=(-b + sqrt(d))/(2.0 *a);  
r2=(-b - sqrt(d))/(2.0 *a);  
printf ("\n Roots are %6.2f and %6.2f ",r1,r2);  
}  
getch();
```

## 'C' Scope

```
}
```

Another example of Library function is `strlen()` which returns the length of the string and `strrev()` which reverses the given string are declared in `string.h` Header File.

The functions that are already declared and defined and are available with the 'C' compiler are known as Library Functions.

Library functions help you in compact and efficient coding as well as in faster development of both simple and complex programs.

```
#include<stdio.h>
main()
{
    char str[20]="scope";
    int i,len;
    for(i=0;str[i];i++);
    for(i--;i>=0;i--)
    {
        printf ("%c",str[i]);
    }
}
```

here is a program that reverses the given string, we can do the same job using library function in more efficient way as

..

```
#include<string.h>
#include<stdio.h>
main()
```

## 'C' Scope

```
{
char str[20]="scope";
printf("%s",strrev(str));
}
```

##### OUTPUT #####

epocs

Now it is very clear that library function helps user a lot in program development.

### **[2] User defined functions**

These functions are defined & made by the user for specific requirements.

```
#include<stdio.h>
main()
{
scope();
jss();
race();
}

scope()
{
printf ("\nSchool For Computer Operator's and
Programmer's Education");
}

jss()
{
```

## 'C' Scope

```
printf (“\nJai Shikshan Sansthan”);  
}  
  
race()  
{  
printf (“\nRajasthan Academy for Computer Education”);  
}
```

The Result of the said program is

School for computer operator’s and programmer’s  
education  
Jai Shikshan Sansthan  
Rajasthan Academy for Computer Education

Here we have four functions where main is the calling function, and is calling three functions named scope (), jss(), race().

In first line scope(), main is calling a function named scope() and transferring the program control to that block and prints the line

School for computer operator’s and programmer’s  
education.

After the complete execution of scope() the control backs to the main(). After this the next line of main() i.e. jss() it takes control to jss() and prints the line

Jai Shikshan Sansthan

Similarly third line

Rajasthan academy of computer education



## 'C' Scope

Will be printed.

```
#include<stdio.h>
main()
{
    scope();
    jss();
}

scope()
{
    printf ("School for computer operator's and programmer's
education ");
    jss();
}

jss()
{
    printf ("Jai Shikshan Sansthan");
}
```

##### OUTPUT #####

School for computer operator's and program's education  
Jai Shikshan Sansthan  
Jai Shikshan Sansthan

Types of user defined functions

- (1) No arguments and no return value
- (2) No argument but with return value
- (3) With arguments and with return value

## 'C' Scope

### (1) No arguments and no return value

The above example illustrates the function with no arguments and no return value.

### (2) No arguments but return value

```
#include<stdio.h>
main()
{
    int a;
    a=sum();
    printf ("%d",a);
}

sum()
{
    int i=10,j=15,k;
    k=i+j;
    return(k);
}
```

result is 25.

In this program main is calling sum function.  
The sum function initializes two variables i and j with value 10 and 15 respectively and stores the sum in k i.e. 25 and the third statement of function sum returns the value of k to the main function, which is stored in the variable a.  
And finally the value of variable a is printed.

### (3) Functions with arguments and return values

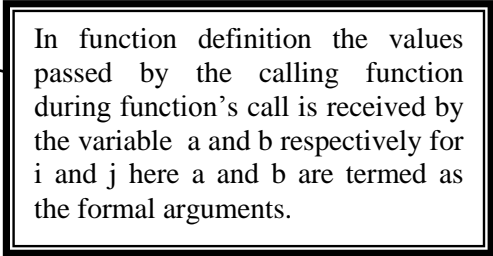
```
#include<stdio.h>
main()
{
    int i=10,j=15,k;
    k=sum(i,j);
    printf ("%d",k);
}
```

i and j are the arguments passed to the function sum from the calling function i.e main function.  
And here it is known *actual arguments*  
**NOTE** : arguments are the values passed along with the function call.

## 'C' Scope

```
}
```

```
sum(int a, int b)
{
    return (a+b);
}
```



In function definition the values passed by the calling function during function's call is received by the variable a and b respectively for i and j here a and b are termed as the formal arguments.

Note: The number and arguments and their types must be same for both actual and formal arguments (parameters)

```
#include<stdio.h>
main()
{
    float a=1.2,b=1.6;
    printf ("%f",sum(a,b));
}
```

```
sum(float x,float y)
{
    return(x+y);
}
```

The above program will produce an error. Because sum () is returning the value that is float. By default every function returns integer value. But we can make function to return values other than integers. For this we need to declare the function before calling and that is known as function prototyping, so the modified form of the above program is

## 'C' Scope

```
#include<stdio.h>
main()
{
    float a=1.2,b=1.6,sum(float,float);
    printf ("%1.1f",sum(a,b));
}
```

```
float sum(float x,float y)
{
    return(x+y);
}
```

##### OUTPUT #####

2.8

Note: it is necessary to declare the functions and it's return type. But if the function is returning integer or character type value it not necessary to write the prototype.

### **Concept of call by value and call by reference**

#### **1. Call By Value**

In call by value a function is called by supplying it to a value.

Ex. 2.

Program to swap two numbers using call by value

```
#include<stdio.h>
main()
{
```

## 'C' Scope

```
int a=10,b=12;
printf (“\nThe Value of a and b before swapping is “);
printf (“\na=%d  b=%d”,a,b);
swap(a,b);
printf (“\nThe Value of a and b after swapping is “);
printf (“\na=%d  b=%d”,a,b);
}
swap(int a, int b)
{
  int c;
  c=a;
  a=b;
  b=c;
  printf (“\nThe Value of a and b in swap function is “);
  printf (“\na=%d  b=%d”,a,b);
}
```

##### OUTPUT #####

The value of a and b before swapping is  
a=10 b=12

The value of and in swap function is  
a=12 b=10

The value of a and b after swapping is  
A=10 b=12

In the above examples in main function the values of a and b are 10 and 12 respectively but in swap function it is 12 and 10 respectively even after the value of a and b remains same i.e 10 and 12 respectively. The reason is in call by value every functions maintains the separate copy of the

## 'C' Scope

variables hence changing the value of variable does not effects the value of variables in other functions.

### **2. Call by reference**

In call by reference we pass the address of the variable instead of value as given below.

Ex. 3.

Program to swap between two numbers using call by reference

```
main()
{
    int a=10,b=12;
    printf("\nThe Value of a and b before swapping is ");
    printf("\na=%d b=%d",a,b);
    swap(&a,&b);
    printf("\nThe Value of a and b after swapping is ");
    printf("\na=%d b=%d",a,b);
}

swap(int *a, int *b)
{
    int c;
    c = *a;
    *a = *b;
    *b = c;
    printf("\nThe Value of a and b in swap function is ");
    printf("\na=%d b=%d",*a,*b);
}
```

## 'C' Scope

##### OUTPUT #####

The result is

The value of a and b before swapping is

a=10 b=12

the value of and in swap function is

a=12 b=10

The value of a and b after swapping is

a=12 b=10

In above program we are passing the address of a and b to swap function and swap functions is receiving the address of these variables through pointers and this causes only one copy of value to be available for both modules and hence changes made at one functions are reflected to other function.

### **Advantage of call by references**

- (1) function can return more than one value at a time
- (2) single copy of variables can be used among many functions.

Ex. 4.

Passing Array to a functions

```
#include<stdio.h>
main()
{
char str[20]="scope";
clrscr();
```

## 'C' Scope

```
upper(str);  
puts(str);  
}
```

```
upper(char *nstr)  
{  
    int i;  
    for(i=0;nstr[i];i++)  
    {  
        nstr[i]=nstr[i]-32;  
    }  
}
```

In above program the character array can be passed to the called functions simply by passing the base address of str. And as we know we can get the address of a variable only in the pointer variable so in called function we use char \*nstr to collect the base address of str.

Ex. 5.

Program to show roman numerals of decimal numbers

```
#include<stdio.h>  
main()  
{  
    int no;  
    clrscr();  
    printf ("Enter the number : ");  
    scanf ("%d",&no);  
    fflush(stdin);  
    yr=roman(no,1000,'m');  
    yr=roman(no,500,'d');  
    yr=roman(no,100,'c');
```



## 'C' Scope

```
yr=roman(no,50,'l');
yr=roman(no,10,'x');
yr=roman(no,5,'v');
yr=roman(no,1,'i');
getch();
}
roman(int y,int k, char ch)
{
int i,j;
if(y==9)
{
printf ("ix");
return(y%9);
}
if(y==4)
{
printf("iv");
return(y%4);
}
j=y/k;
for(i=1;i<=j;i++)
{
printf ("%c",ch);
}
return(y-k*j);
}
```

##### OUTPUT #####

Enter a Number : 500

**USES**

## 'C' Scope

- (1) It helps you in top down modular programming i.e. a complex problem can be divided into simple modules.
- (2) It makes testing and debugging very easy and fast.
- (3) The repeated statements in a program can be avoided by use of functions which makes program code compact.
- (4) A function can be used by many other programs.

## **RECURSION**

In previous section you have learned about the functions. But what happened when a function call itself?

When a function calls itself then this is known as recursion. In mathematics and computer science the meaning of recursion is self reference.

In recursion actually what happens is that the functions has an *anchor statement* that again and again calls itself & this calling's termination or in action depends upon a particular condition.

Here a stack is allocated to the recursive function, every time it calls itself the statement is pushed into the stack. When the termination condition is met the items are popped out of the stack & actions according to that are performed.

Let's see an example to this...

Recursion can be explained by the following example.

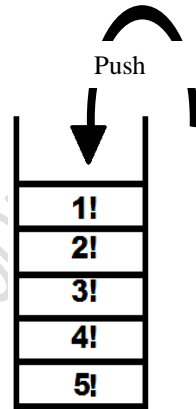
Ex. 6.

Program to factorial of a given number using recursion

## 'C' Scope

```
#include <stdio.h>
main()
{
    int n;
    clrscr();
    printf ("Enter a value for factorial : ");
    scanf ("%d",&n);
    printf ("%d",fact(n));
    getch();
}

fact (int n)
{
    int x=1;
    if (n==0)
        return 1;
    else
        return(n*fact(n-1));
}
```



##### OUTPUT #####

Enter a value for factorial : 5  
120

We know that for every positive integer the factorial of n is  $n! = n*(n-1)!$

Step 1. to compute 5!, 5 will be passed by the main function to factorial function and in factorial function the computation of 5! will be sent to the stack and it will try to compute 4!.

Step 2. In this step to compute 4! it will send the due work to the stack and try to compute the 3!

## 'C' Scope

Step 3. In this step to compute 3! it will send the due work to the stack and try to compute the 2!

Step 4. In this step to compute 2! it will send the due work to the stack and try to compute the 1!

Step 5. In this step to compute 1! it will send the due work to the stack and try to compute the 0!

Step 6. In this step to compute 0! it is equals to 1.

Step 7. It will be returned to the stack and compute the 1! using the value of 0!.

Step 8. Will be returned to the next due work in stack and compute 2! using the value of 1!

Step 9. Will be returned to the next due work in stack and compute 3! using the value of 2!

Step 8. Will be returned to the next due work in stack and compute 4! using the value of 3!

Step 8. Will be returned to the next due work in stack and compute 5! using the value of 4!

Step 9. Finally the 5! will be calculated and returned to the original function (main) and print the 5! value.

Ex. 7.

To find the Greatest Common divisor using Euclid's Algorithm with recursion

```
gcd(int a,int b)
{
    if((a>=b) && ((a%b)==0))
        return(b);
    else
        gcd(a,(a%b));
}
void main()
{
```

## 'C' Scope

```
int a,b,res;
clrscr();
printf("Enter the first number \n");
scanf("%d",&a);
printf("\n Enter the second number \n");
scanf("%d",&b);
res=gcd(a,b);
printf("\n Greatest Common Divisor is : %d",res);
getch();
}
##### OUTPUT #####
```

Enter the first number  
10

Enter the second number  
15

Greatest Common Divisor is : 10

***A function is called recursive if it calls it self or calls one function that calls another and so on until the first is called again.***

### **Types of recursions**

- a. Preemptive recursive function.
- b. Non-Preemptive recursive function.

Ackerman's function (non-preemptive)

## 'C' Scope

Let F is a function

F(a,b)

b=1 if a=0

F(a-1,1) if b=0

else

F(a-1,a(a,b-1))

When a function call itself as an argument to itself. This is Ackerman's function and is a very typical situation of recursion.

Ex. 8.

Program to compute number using recursion

```
int fibo(int num)
{
    if (num <=1)
        return num;
    else
        return (fibo(n-1) + fibo(n-2));
}
```

### **Tower of Hanoi Game**

This is the very famous example of recursion. This game consists of three polls and a set of disks of increasing size.

Initially all the disks are available in first poll, and you have to move all the disks from the poll a to poll c, with the help of auxiliary poll b.

Rules

## 'C' Scope

1. You can move only one disk at a time.
2. You can move the disk from the top only
3. No bigger disk can be placed on to the smaller disk

Your program should ask for the number of disks, and display the sequence of replacement.

If numbers of disks are four, then it will take 15 replacements.

Now try to play this game manually and get three polls and 64 disks, and start replacement manually and contact me when you get the success to move all the disks following all the rules of this game. I will wait for you.

This process will not take a long time it will take only around 8760000000000000 hours or 1000000000000 Years to be completed.

So friends, please try it.....

Ex. 9.

Program to implement Tower of Hanoi

```
#include <stdio.h>
main()
{
    int num;
    clrscr();
    printf ("Enter number or disks : ");
    scanf ("%d",&num);
    fflush(stdin);
```

## 'C' Scope

```
hanoi('a','c','b',num);  
}
```

```
hanoi(char a,char b,char c,int num)  
{  
if (num==1)  
printf ("Move disk from %c to %c \n",a,b);  
else  
{  
hanoi(a,c,b,num-1);  
hanoi(a,b,c,1);  
hanoi(c,b,a,num-1);  
}  
}
```

##### OUTPUT #####

Enter number or disks :

1. Move disk from a to b
2. Move disk from a to c
3. Move disk from b to c
4. Move disk from a to b
5. Move disk from c to a
6. Move disk from c to b
7. Move disk from a to b
8. Move disk from a to c
9. Move disk from b to c
10. Move disk from b to a
11. Move disk from c to a
12. Move disk from b to c
13. Move disk from a to b
14. Move disk from a to c
15. Move disk from b to c



## 'C' Scope

Ex. 10.

Program to reverse individual characters of line of string

```
#include <stdio.h>
main()
{
    clrscr();
    printf ("Enter a line of text :");
    rev();
}

rev()
{
    char ch;
    if ((ch=getchar())!='\n')
        rev();
    putchar (ch);
    return;
}
```

##### OUTPUT #####

Enter a line of text :This book is written by Nishat

tahsiN yb nettirw si koob sihT

Ex. 11.

Program to reverse the Name order

```
#include <stdio.h>
char *name[]={
```

## 'C' Scope

```
"Nishat",  
"Raman",  
"Vaneja",  
"Ranjan",  
"Vishal",  
};
```

##### OUTPUT #####

```
Vishal  
Ranjan  
Vaneja  
Raman  
Nishat
```

```
main()  
{  
clrscr();  
display(name);  
return(0);  
}  
  
display ( char **nam)  
{  
if (*nam != 0)  
{  
display(nam +1);  
printf ("\n%s",*nam);  
}  
}
```

Ex. 12.

Program to convert decimal number into Binary

## 'C' Scope

```
#include <stdio.h>
main()
{
    int num;
    clrscr();
    printf ("Enter a number : ");
    scanf ("%d",&num);
    binary(num);
    getch();
}
```

```
binary (int num)
{
    if (num<=0)
        return;
    else
    {
        binary(num /2);
        printf ("%d",num%2);
    }
}
```

##### OUTPUT #####

```
Enter a number : 75
1001011
```

### **Depth of Recursion:**

Let F is a recursive function. When F is called it is assigned by a level number 1 and each time it is called recursive

## **'C' Scope**

level number increases by one. So number of levels is one more than level of execution.

The depth of recursion refers to the maximum level number of F during its execution.

### **Summary**

- Function can return one value at a time, but a function can have more than one statement using conditional constructs.
- Functions should return a value but if a function is not returning any value is called as void function and must be declared with keyword void as a return type.
- Any function can be called any number of times
- Every 'C' program must have one main function.
- Functions prototyping is must in case of functions returning values other than int and char type.
- The position of main() can be any where in the program.
- Every programme must have one and only one main()
- During function call with arguments the number and data types in calling and called functions must be the same.
- Every time when the function calls itself it must be closer the condition for which it comes out.
- A recursive function must not be continuing infinitely, there must be a condition, for which the function does not call itself.
- It is used to solve the complex problems easily that would be difficult using iterative method (loops).

## 'C' Scope

---

### Self Review

- Q1. Write functions to implement the following without using built-in functions?
- Finding Length of string
  - Comparing two strings
  - Checking whether the string is palindrome
  - Swapping two strings
  - Incrementing alphabets of strings
  - Converting a string to lower & upper case
- Q2. Write a user defined function to implement 8-queen problem using recursion?
- Q3. Write a program to quicksort using loops as well as recursion?
- Q4. Write a program to implement Breadth First Search & Depth First Search?
- Q5. Write a program to implement binary search using recursion?
- Q6. Write a program to find power of a number using recursive function?
- Q7. Write a program to write a recursive function heap sort?

# CHAPTER - 8

## Storage class specifiers

Every variable has got a storage specifier which tells :

- (1) where the data will be stored (ram or cpu registers)
- (2) what will be it's life and
- (3) what will be it's default initial value.
- (4) What is the scope of the variable.
- (5) Whether the variable has any external linkage or not.
- (6) Whether the variable is statically available or automatically available for use during execution.

'C' provides four types of storage class specifiers. That are used to refine the declaration of a variable, a function, and parameters.

### **Life of a variable:**

#### 1. Auto

A variable having automatic storage is deleted when the block in which it was declared exits.

You can only apply the auto storage class specifier to names of variables declared in a block or to names of function parameters. The storage class specifier auto is usually redundant in a data declaration.

## 'C' Scope

By default all the data types are treated as auto class specifiers and

- (1) The default initial value is junk value (Garbage)
- (2) Always stored in memory (RAM)
- (3) Life is till the program control is in the block in which variable is declared.
- (4) (Scope) - In the block where it is defined.

Ex. 1.

Program to swap numbers using Call by Value Procedure

```
#include<stdio.h>
main()
{
    auto int i,j,k;
    i=10;
    j=12;
    swap(i,j);
    printf ("the values of i and j in main function are i=%d
j=%d",i,j);
}

swap(int i,int j)
{
    int k;
    k=i;
    i=j;
    j=k;
    printf ("the values of i and j in swap function are i = %d
j=%d",i,j);
}
```

## 'C' Scope

---

##### OUTPUT #####

The values of i and j in main functions are i=10 j=12

The values of i and j in swap functions are i=12 j=10

So the values of i and j are different in main and swap hence we can say that the scope of auto storage class specifiers is limited to the block where there are declared.

### 2. Register

The register storage class specifier indicates to the compiler that the value of the object should reside in a machine register. The compiler does not honor this request.

It occupies limited size because number of registers available on most systems, variables can actually be placed in registers.

If the compiler does not allocate a machine register for a register object, the object is treated as having the storage class specifier auto because we are only requesting not forcing the compiler to store the variable or object in CPU register. A register storage class specifier shows that the variable, such as a loop control variable, that is heavily used and that the programmer hopes to enhance performance by minimizing access time and also the execution time.

Let's work on one of its example...



## 'C' Scope

Ex. 2.

Program to implement register specifier

```
#include<stdio.h>
void main()
{
    register int i;
    clrscr();
    for(i=0;i<10;i++)
        printf("%d ",i);
    getch();
}
```

##### OUTPUT #####

0 1 2 3 4 5 6 7 8 9

### 3. Static

Variables declared with static storage class specifier have static storage, that is only once the memory is allocated to these objects when program's execution begins & memory is freed only when the program terminates.

Here the scope of static variable is local to the block. The static variable once initialized retains the value & manipulations in that value till the program are terminated.

Let us see a program:-

Ex. 3.

Write program to compare between static and auto storage class specifier.

## 'C' Scope

```
void mystring(char *str)
{
    static int i=0;
    int j;
    printf("\n");
    for(j=i;j<strlen(str);j++)
        printf("%c ",str[j]);
    i++;
}
void main()
{
    char *str;
    clrscr();
    printf("\n Enter the string \n");
    scanf("%s",str);
    mystring(str);
    mystring(str);
    mystring(str);
    mystring(str);
    getch();
}
```

##### OUTPUT #####

Enter the string  
scope

s c o p e  
c o p e  
o p e  
p e

if the above program written  
without using static keyword the

scenario would have been  
like this.

```
void mystring(char *str)
{
    int i=0;
    int j;
    printf("\n");
    for(j=i;j<strlen(str);j++)
        printf("%c ",str[j]);
    i++;
}
void main()
{
    char *str;
    clrscr();
    printf("\n Enter the string \n");
    scanf("%s",str);
    mystring(str);
    mystring(str);
    mystring(str);
    mystring(str);
    getch();
}
```

##### OUTPUT #####

Enter the string  
scope

s c o p e  
s c o p e  
s c o p e  
s c o p e

## 'C' Scope

---

In the first program that used static variable *i*, the value of *i*, is retained throughout the program that again & again when *i* is incremented the value of *i* increases with respect to the previous value of *i* & not initialized to a value previously initialization used i.e it retain the value of *i* & every time when *j* is given the value of *i* for loop shifts the string printing by one place.

On the other hand if the variable *i* is not declared as static then every time when the function is called, every time the value of *i* is set/initialized to value 0(zero). So every time the for loop prints the whole string & also its increment statement part does not proves to be fruitful because every time a new local variable is assigned a new memory that is for *i* , & auto variables do not retain the value in memory if it is not in use i.e they are freed when the block is freed, & are reassigned when block/function is called for.

### **4. Extern**

The variable declared using extern storage class specifier are stored in memory with default zero initial value and continue to stay within the memory until the execution of the program is not terminated.

Variables declared as extern can be accessed by all functions in the program, which is they act like global variable that can be accessed & used by all blocks in the program, thus avoiding unnecessary passing of these variables as arguments during function call.

## 'C' Scope

Variables declared outside any function definition are treated as variables with extern storage class (can also be said as global).

Using extern as storage specifier also enables the user to take many functions from outside the program or any other file, thus increasing the **degree of code reuse**.

Let us see its program in action...

Ex. 4.

Program to show use of extern keyword

```
#include<stdio.h>
extern int i=5;
void func()
{
    printf("inside func i = %d",i);
}
void main()
{
    clrscr();
    func();
    i++;
    printf("\n inside main i = %d",i);
    getch();
}
```

##### OUTPUT #####

```
inside func i = 5
inside main i = 6
```

## 'C' Scope

In the above program i is declared as extern variable whose scope & life persists till the end of the program. Here the extern variable i , act as global & its scope remains throughout the program, that is why it can be accessed anywhere in the program.

## Summary

- 'C' provides four types of storage class specifiers. That are used to refine the declaration of a variable, a function, and parameters.
- You can only apply the auto storage class specifier to names of variables declared in a block or to names of function parameters. The storage class specifier auto is usually redundant in a data declaration.
- By default all the data types are treated as auto class specifiers and the default initial value is junk value (Garbage). It is always stored in memory (RAM) & its life is till the program control is in the block in which variable is declared.
- The register storage class specifier indicates to the compiler that the value of the object should reside in a machine register keep in mind that we are only requesting not forcing the compiler to store the variable or object in CPU register.
- Variables with static storage class specifier have static storage, that is only once the memory is allocated to these objects when program's execution begins & memory is freed only when the program terminates.
- The variable declared using extern storage class specifier are stored in memory with default zero

## 'C' Scope

initial value and continue to stay within the memory until the execution of the program is not terminated.

- Using extern as storage specifier also enables the user to take many functions from outside the program or any other file, thus increasing the **degree of code reuse**.

### Self Review

- Q1. What do you understand by scope & life of a variable? What is the difference between two?
- Q2. Explain the necessity of storage class specifiers?
- Q3. What happens when variable is specified as of register storage class & CPU doesn't find free registers?
- Q4. Explain with an example how extern class storage specifier affects degree of code reuse?

# **Chapter – 9**

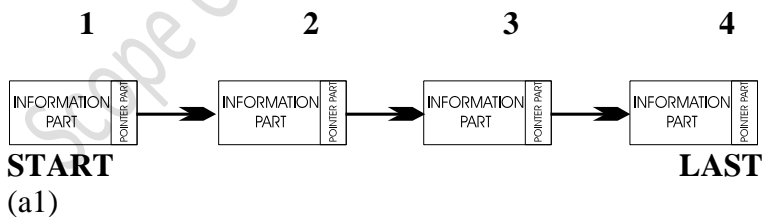
## **LINKED LIST**

In previous chapters we have learned about arrays (fixed length memory storage), this method of memory allocation can be used for certain applications you have used previously, but there are certain other operations where the use of this linear data structure cannot be efficient.

When we write a program we can not exactly decide the amount of data storage, because it often depends on the particular data being processed.

The linked-allocation method of storage can result in both the efficient use of computer storage and time.

Linked list consists of nodes, and a node is consists of two parts the **INFO** part and **POINTER** part (pointer means address to the next node)

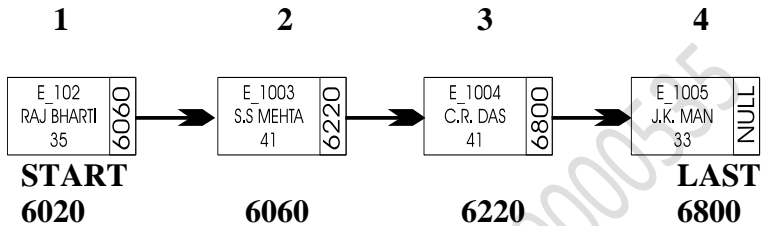


Consider the list (a1) which contains the four nodes. First node 1 can be called as start and the node number 4 can be called as the last. The pointer part of node 1 is containing the address of node number 2 and the pointer part of the

## 'C' Scope

node 2 is pointing to the node 3 .... . The pointer part of node 3 is pointing to **NULL**. So in this way it forms a linear linked list.

Now suppose we have EMP\_CODE, NAME and AGE



In the above linked list node no. 1 is the START and node no. 2 is residing at address 6060 so the address part of node no. 1 will point to the address 6060 and in this way the address part of node number 2 will point to 6220 which is the address of address node number 3, but the address part of node no. 4 will point to the NULL, because there is no node existing beyond this node.

### SINGLY UNSORTED LINKED LIST

We can insert a node in singly unsorted linked list.

Ex. 1.

Program for Insertion singly unsorted linked list

```
#include<stdio.h>
struct list {
    char name[20];
    int age;
    struct list *next;
};
```



## 'C' Scope

```
struct list
*start='\0', *last='\0', *newnode, *pt, *makenode(char *,int);
main()
{
char nm[20];
int i,j,k,ag;
clrscr();
while(1)
{
printf ("Enter the name : ");
gets(nm);
fflush(stdin);
if((strlen(nm))==0)
break;
printf ("Enter the age : ");
scanf("%d",&ag);
fflush(stdin);
newnode=makenode(nm,ag);
if(!start)
{
start=last=newnode;
}
else
{
last->next=newnode;
last=newnode;
}
}

for(pt=start;pt;pt=pt->next)
{
printf ("% -20s %d\n",pt->name,pt->age);
```

## 'C' Scope

```
    }
    getch()
}
struct list *makenode(char *nm,int ag)
{
    struct list *ptr;
    ptr=(struct list *)malloc(sizeof(struct list));
    strcpy(ptr->name,nm);
    ptr->age=ag;
    ptr->next='\0';
    return(ptr);
}
```

### DOUBLY UNSORTED LINKED LIST

Ex. 2.

Program for insertion in doubly unsorted linked list

```
#include<stdio.h>
struct list {
    char name[20];
    int age;
    struct list *next;
    struct list *pre;
};

struct list *start='\0', *last='\0', *newnode, *ptr,
*makenode(char *,int);
main()
{
    char nm[20];
    int i,j,k,ag;
    clrscr();
```

## 'C' Scope

```
while(1)
{
    printf ("Enter the name : ");
    gets(nm);
    fflush(stdin);
    if((strlen(nm))==0)
        break;
    printf ("Enter the age : ");
    scanf("%d",&ag);
    fflush(stdin);
    newnode=makenode(nm,ag);
    if(!start)
    {
        start=last=newnode;
    }
    else
    {
        last->next=newnode;
        newnode->pre=last;
        last=newnode;
    }
}
for(ptr=start;ptr;ptr=ptr->next)
{
    printf ("% -20s %d\n",ptr->name,ptr->age);
}
getch();

printf ("REVERSE LIST : \n");
for(ptr=last;ptr;ptr=ptr->pre)
{
    printf ("% -20s %d\n",ptr->name,ptr->age);
```

## 'C' Scope

```
    }
    getch();
}
struct list *makenode(char *nm,int ag)
{
    struct list *ptr;
    ptr=(struct list *)malloc(sizeof(struct list));
    strcpy(ptr->name,nm);
    ptr->age=ag;
    ptr->next='\0';
    ptr->pre='\0';
    return(ptr);
}
```

In doubly linked list there are two pointers first point the previous and second points to the next node  
We can insert a node in doubly unsorted list.

Ex. 3.

Program for insertion in sorted doubly linked list

```
#include<stdio.h>
struct list {
    char name[20];
    int age;
    struct list *next;
    struct list *pre;
};
struct list
*start='\0',*last='\0',*newnode,*ptr,*makenode(char *,int);
main()
{
    char nm[20];
```

## 'C' Scope

```
int i,j,k,ag;
clrscr();
while(1)
{
printf ("Enter the name : ");
gets(nm);
fflush(stdin);
if((strlen(nm))==0)
break;
printf ("Enter the age : ");
scanf("%d",&ag);
fflush(stdin);
newnode=makenode(nm,ag);
if(!start)
{
start=last=newnode;
}
else
{
for(ptr=start;strcmp(ptr->name,nm)<0 && ptr;ptr=ptr-
>next);
if(ptr)
{
if(ptr==start)
{
start->pre=newnode;
newnode->next=start;
start=newnode;
}
else
{
newnode->pre=ptr->pre;
ptr->pre->next=newnode;
```

## 'C' Scope

```
    newnode->next=ptr;
    ptr->pre=newnode;
}
}
else
{
    last->next=newnode;
    newnode->pre=last;
    last=newnode;
}
}
}
for(ptr=start;ptr;ptr=ptr->next)
{
    printf ("%s %d\n",ptr->name,ptr->age);
}
getch();
}
```

```
struct list *makenode(char *nm,int ag)
{
    struct list *ptr;
    ptr=(struct list *)malloc(sizeof(struct list));
    strcpy(ptr->name,nm);
    ptr->age=ag;
    ptr->next='\0';
    ptr->pre='\0';

    return(ptr);
}
```

## 'C' Scope

Ex. 4.

Program for deletion in doubly linked list

```
#include<stdio.h>
struct list {
    char name[20];
    int age;
    struct list *next;
    struct list *pre;
    int flag=0;
};
struct list *start='\0', *last='\0', *newnode, *ptr,
*makenode(char *,int);
main()
{
    char nm[20],dname[20];
    int i,j,k,ag;
    clrscr();
    while(1)
    {
        printf ("Enter the name : ");
        gets(nm);
        fflush(stdin);
        if((strlen(nm))==0)
            break;
        printf ("Enter the age : ");
        scanf("%d",&ag);
        fflush(stdin);
        newnode=makenode(nm,ag);
        if(!start)
        {
            start=last=newnode;
        }
    }
}
```

## 'C' Scope

```
else
{
    for(ptr=start;strcmp(ptr->name,nm)<0 && ptr;ptr=ptr-
>next);
    if(ptr)
    {
        if(ptr==start)
        {
            start->pre=newnode;
            newnode->next=start;
            start=newnode;
        }
        else
        {
            newnode->pre=ptr->pre;
            ptr->pre->next=newnode;
            newnode->next=ptr;
            ptr->pre=newnode;
        }
    }
    else
    {
        last->next=newnode;
        newnode->pre=last;
        last=newnode;
    }
}
for(ptr=start;ptr;ptr=ptr->next)
{
    printf ("%s %d\n",ptr->name,ptr->age);
}
getch();
```



## 'C' Scope

```
printf ("Enter the Name to Be Deleted : ");
gets(dname);
fflush(stdin);
for(ptr=start;strcmp(ptr->name,dname)!=0 &&
ptr;ptr=ptr->next);
if(ptr)
{
if(ptr==start)
{
start=start->next;
ptr->next='\0';
}
else if (ptr==last)
{
last=last->pre;
last->next='\0';
}
else
{
ptr->pre->next=ptr->next;
ptr->next->pre=ptr->pre;
ptr->pre='\0';
ptr->next='\0';
}
}
else
{
printf ("Sorry Record Not Found :\n");
}

for(ptr=start;ptr;ptr=ptr->next)
{
```

## 'C' Scope

```
    printf ("%s %d\n",ptr->name,ptr->age);
    }
}
struct list *makenode(char *nm,int ag)
{
    struct list *ptr;
    ptr=(struct list *)malloc(sizeof(struct list));
    strcpy(ptr->name,nm);
    ptr->age=ag;
    ptr->next='\0';
    ptr->pre='\0';
    return(ptr);
}
```

Ex. 5.

Program for insertion in sorted singly linked list

```
#include<stdio.h>
struct list {
    char name[20];
    int age;
    struct list *next;
};
struct list *start='\0', *last='\0', *pptr, *newnode, *ptr,
*makenode(char *,int);
main()
{
    char nm[20];
    int i,j,k,ag;
    clrscr();
    while(1)
    {
        printf ("Enter the name : ");
```

## 'C' Scope

```
gets(nm);
fflush(stdin);
if((strlen(nm))==0)
    break;
printf ("Enter the age : ");
scanf("%d",&ag);
fflush(stdin);
newnode=makenode(nm,ag);
if(!start)
{
    start=last=newnode;
}
else
{
    for(ptr=pptr=start;strcmp(ptr->name,nm)<0 &&
ptr;pptr=ptr,ptr=ptr->next);
    if(ptr)
    {
        if(ptr==start)
        {
            newnode->next=start;
            start=newnode;
        }
        else
        {
            pptr->next=newnode;
            newnode->next=ptr;
        }
    }
    else
    {
        last->next=newnode;
        last=newnode;
    }
}
```

## 'C' Scope

---

```
    }  
    }  
    }  
    for(ptr=start;ptr;ptr=ptr->next)  
    {  
        printf ("%s %d\n",ptr->name,ptr->age);  
    }  
    getch();  
}
```

```
struct list *makenode(char *nm,int ag)  
{  
    struct list *ptr;  
    ptr=(struct list *)malloc(sizeof(struct list));  
    strcpy(ptr->name,nm);  
    ptr->age=ag;  
    ptr->next='\0';  
    return(ptr);  
}
```

## Summary

- When we write a program we can not exactly decide the amount of data storage, because it often depends on the particular data being processed, so for this purpose Linked Lists are used.
- This representation results in both the efficient use of computer storage and time.
- Linked list consists of nodes, and a node is consists of two parts the **INFO** part and **POINTER** part.

## Self Review

## 'C' Scope

- Q1. Write a program that takes the values from the user for each node and then sort the linked list?
- Q2. Write a program to implement graph, insertion deletion & graph traversal i.e BFS & DFS implementation?
- Q3. Write a program to insert element in a sorted Linked List?
- Q4. Write a program to implement following using linked list :-
- Stack
  - Queue
  - Doubly Linked List
  - Tree
- Q5. Write a program to implement priority queue (and Heap) using linked list ?

# Chapter 10

## FILE HANDLING

Consider the statement

```
FILE *fptr;
```

```
fptr = fopen("SCOPE.TXT","r");
```

Here fptr is a pointer requesting the operating system to open a file scope.txt for reading purpose. If scope.txt is not existing then a null value will be returned to the fptr.

For another example

```
fptr1 = fopen("ABC.TXT","w");
```

Here fptr1 is a pointer requesting the operating system to open a file for writing purpose. For this abc.txt must not be an existing file. If it is existing then it's contents will be overwritten.

“r” and “w” are the file opening modes. All the file opening modes are described with their purpose and action in following table.

## 'C' Scope

<b>Mode</b>	<b>Purpose and Action</b>
r	<b>To open a File in Read Mode</b> In read mode A file must exist, if it is not existing a null will be returned
w	<b>To open a file in Write Mode</b> In write mode if a file already exist then its contents will be overwritten. If It does not exist a new file will be created
r+	<b>To open a file for reading and writing</b> In r+ mode the file must already exist
w+	<b>To open a file for reading and writing</b> In w+ mode if the file exists its contents will be overwritten.
a	<b>To open a file to append.</b> In this mode if the file is existing data will be added at the end of it. If it is not existing it will be created.
a+	<b>To open a file for reading and appending</b> To add the text at the end of the text if it is not existing it will be created.
rb	To open a binary file for reading
r+b	To open a binary file for read and

## 'C' Scope

	write
wb	To open a binary for the purpose of writing
w+b	To create a binary file for reading and writing
ab	To open a file to append in binary mode.
a+b	To append or create binary file for reading and writing.

We can better discuss about this through following program segment.

### Tracing End Of File

We tried to incorporate reading from & writing into the file but what if during reading the file end of file is reached, as 'C' file pointer is not intelligent enough to end reading the file when there is no more content in the stream, it continues to read end of file from the stream. To convince you on this lets see an example

Ex. 1.

Program to illustrate use of file pointer pointing functions

```
#include<stdio.h>
void main()
{
    FILE *fp;
    int i;
    char ch;
    clrscr();
```



## 'C' Scope

```
fp=fopen("jaya.txt","r");
printf("\n Now pointer points at %d",ftell(fp));

fseek(fp,1,2);
i=ftell(fp);
ch=getc(fp);

printf("%ld %c",i,ch);
i=ftell(fp);
ch=getc(fp);

printf("%ld %c",i,ch);
i=ftell(fp);
ch=getc(fp);

printf("%ld %c",i,ch);
i=ftell(fp);
ch=getc(fp);

printf("%ld %c",i,ch);
getch();
}
```

#####OUTPUT#####

Now pointer points at 0 65 65 65 65

In the above program if we are making the file pointer to reach beyond the EOF, the file pointer \*fptr and read characters, but the pointer remains at EOF location, keeps on reading on that location & doesn't go beyond that.

## 'C' Scope

To avoid the situation of reading the EOF again & again during a loop we can give a condition !EOF reaching or can use a function feof().

### File I/O

The input and output i.e reading from a file & writing into the file differs from ordinary input & output statements that we use in 'C', here the file pointer comes into play( that points to the next memory location to be read or written upon). There are different type of I/O statements for streams, these are

### fgetc() & fputc()

fgetc() reads a single character from the file that is pointed by file pointer, its syntax can be

```
char ch = fgetc(FILE *ptr)
```

Lets code an example program for this

Ex. 2.

Program for reading characters From a file

```
#include <stdio.h>
main()
{
    char ch;
    FILE *fr;
    fr=fopen("scope","r");
    if (!fr)
    {
        printf("Can not open a File. ... ");
    }
}
```

## 'C' Scope

```
exit();
}
ch=getc(fr)
while (ch!=eof)
{
    printf ("%c",ch);
    ch=getc(fp);
}
fclose(fp);
}
```

In above program, we have used a FILE pointer 'fr' to open a file in read mode. To open a file we used a function fopen(). If there is any problem to open an existing file or it does not find the file 'scope.txt' then a null value will be returned to 'fr' and it will come out (the program will terminate without doing anything ).

After opening a file "scope.txt" in read mode we have a function getc() which reads a character from file and position to the next character. In this way we can read and print all the character of a file.

At the end of program we have used a function to close the file fclose().

Similarly we can write a program to get the characters from standard input and put them to the file

fputc() whereas writes a single character on the file to which file pointer is pointing on. Its syntax can be written as

## 'C' Scope

```
fputc(char ch, FILE *ptr)
```

example

```
char ch = 'j';  
fputc(ch,fptr);
```

Lets code an example program for this

Ex. 3.

Reading characters from standard input & writing them to a file

```
#include <stdio.h>  
main()  
{  
char ch;  
FILE *fp;  
fp=fopen("scope.dat","w");  
if (!fp)  
{  
printf ("Error opening a file : ");  
exit();  
}  
else  
{  
ch=getchar();  
while(ch !='\n')  
{  
putc(ch,fp);  
ch=getchar();  
}  
fclose(fp);  
}  
getch();
```

## 'C' Scope

}

### String Based file input output

To read and write strings from/in a file we use `fgets()` and `fputs()` functions respectively. It works similarly as `fgetc()` & `fputc()` but rather than dealing with single characters it deals with strings reading & writing.

The syntax of `fputs()` is

```
fputs(word,stream);
```

and the syntax of `fgets()` is

```
fgets(word,n,stream)
```

here `n` is the maximum number of characters can be stored in characters string.

Ex. 4.

Program to demonstrate string reading & writing with files.

```
#include <stdio.h>
main()
{
FILE *fp;
char word[50];
fp=fopen("scope.txt","w");
if (!fp)
{
printf ("Can't Open a file Scope.txt ");
exit();
}
while(1)
{
gets(word);
```

## 'C' Scope

```
if (strlen(word)==0)
    break;
fputs(word,fp);
}
fclose(fp);
}
```

### /\* String reading program demonstration. \*/

```
#include <stdio.h>
main()
{
    FILE *fp;
    char word[50];
    fp=fopen("scope.txt","r");
    if (!fp)
    {
        printf ("Can't Open a file Scope.txt ");
        exit();
    }
    while(1)
    {
        if((fgets(word,50,fp))==NULL)
            break;
        printf ("%s",word);
    }
    fclose(fp);
}
```

Till now we only tried to put the strings or characters in the file, what if we want to put any float, integer or any character or string in the file . For this need to be fulfilled 'C' provides us handsome amount of statements, these are:-

## 'C' Scope

### fread() and fwrite() statements

These statements let the user, enter a structured record into a file, also let the user retrieve the data in the file as a record or we can say tuples.

Let us code an example for this.

```
#include<stdio.h>
void main()
{
    FILE *fp;
    int ch=1;
    struct student
    {
        int rollno;
        char name[15];
        float percentage;
    }s;
    clrscr();
    fp=fopen("STUDENT.DAT","w");
    while(ch!=0)
    {
        printf("\n Enter the value of rollno : ");
        scanf("%d",&s.rollno);
        printf("\n Enter the value of name : ");
        scanf("%s",s.name);
        printf("\n Enter the value of percentage : ");
        scanf("%f",&s.percentage);
        fwrite(&s,sizeof(s),1,fp);
        printf("Wanna add another record, Press
Y=1/N=0");
        scanf("%d",&ch);
    }
}
```

## 'C' Scope

```
        if(ch!=1) break;
    }
    fclose(fp);

    fp=fopen("STUDENT.DAT","r");
    while(fread(&s,sizeof(s),1,fp)==1)
        printf("\n Rollno = %d Name = %s Having percentage
= %f",s.rollno, s.name, s.percentage);
    fclose(fp);
    getch();
}
```

#####OUTPUT#####

Enter the value of name : kalpana  
Enter the value of percentage : 50.00  
Wanna add another record, Press Y=1/N=0 : 1

Enter the value of rollno : 3  
Enter the value of name : mukesh  
Enter the value of percentage : 78.00

Wanna add another record, Press Y=1/N=0 : 0  
Rollno = 2 Name = kalpana Having percentage =  
50.000000  
Rollno = 3 Name = mukesh Having percentage =  
78.000000

### fscanf() and fprintf()

If we want to save any formatted input to be entered in any file. We can use fprintf() to enter any formatted input in any data file. And fscanf() to read the contents of file.



## 'C' Scope

This works exactly like the inbuilt syntaxes of printf() writing on the console screen & scanf() reading from the console screen.

```
#include<stdio.h>
void main()
{
    FILE *fp;
    int ch=1,count=1;
    int r;float p;char n[15];
    clrscr();
    fp=fopen("STUDENT.DAT","w");
    printf("\n Enter the rollno : ");
    scanf("%d",&r);
    printf("\n Enter the name : ");
    scanf("%s",n);
    printf("\n Enter percentage : ");
    scanf("%f",&p);
    fprintf(fp,"%d %s %f",r,n,p);
    fclose(fp);

    fp=fopen("STUDENT.DAT","r");
    fscanf(fp,"%d %s %f",&r,&n,&p);
    printf("\n Rollno = %d Name = %s Having percentage
= %f",r,n,p);
    fclose(fp);
    getch();
}
#####OUTPUT#####
```

Enter the rollno : 1

Enter the name : jaya

## 'C' Scope

Enter percentage : 90.00

Rollno = 1 Name = jaya Having percentage = 90.000000

That was all about the input & output statements but what if we want to trace the position of file pointer when we want to append records, or tell the file pointer to reach any particular location for the purpose of appending any record.

To do the above mentioned task we have two builtin functions named ftell() and fseek() that works respectively.

ftell()

This function return the current position of the file position pointer. The value is counted from the beginning of the file.

```
long ftell (file * fptr);
```

fseek()

Sets the position of file pointer indicator that is associated with the file to a new position.

```
fseek(FILE *ptr, long int offset, int origin)
```

where,

\*ptr is the file pointer associated with the given file.

Offset is how many bytes backward or forward from the origin has to be moved

## 'C' Scope

Origin shows from where to start, it can be of three types:-

SEEK_SET	0	Seek from the start of the file
SEEK_CUR	1	Seek from the current location
SEEK_END	2	Seek from the end of the file

fseek() returns zero upon success, non-zero on failure. You can use fseek() to move beyond a file, but not before the beginning.

Let us see an example to that

Ex. 5.

Program to show implementation of

```
#include<stdio.h>
void main()
{
    FILE *fp;
    int i,roll;
    char name[10];
    float per;
    char ch;
    clrscr();
    fp=fopen("jaya.txt","w");
    printf("\n Enter the records \n");
    for(i=0;i<2;i++)
    {
        scanf("%d %s %f",&roll,name,&per);
        fprintf(fp,"%d %s %f",roll,name,per);
    }
    fclose(fp);
}
```

## 'C' Scope

```
fp=fopen("jaya.txt","a");
printf("\n Now pointer points at %d we are pointing it
at end of the file for appending more records ",ftell(fp));
fseek(fp,+1,2);
for(i=0;i<2;i++)
{
    scanf("%d %s %f",&roll,name,&per);
    fprintf(fp,"%d %s %f",roll,name,per);
}
fclose(fp);
fp=fopen("jaya.txt","r");
while(!feof(fp))
{
    ch=getc(fp);
    printf("%c",ch);
}
getch();
}
```

#####OUTPUT#####

Enter the records

```
3
jaya
90.00

4
kaya
80.00
```

Now pointer points at 0 we are pointing it at end of the file for appending more records

## 'C' Scope

5

maya

70.00

6

haya

60.00

3 jaya 90.000000

4 kaya 80.000000

5 maya 70.000000

6 haya 60.000000

In the above program we are first using `fprintf` to print two records in the file and close the writing operation. Now again we are opening the same file in append mode, as we have discussed previously that file ptr always points the first location record in file, & employing `ftell` proves it in the above program. Now if we want to append then we need to start writing from the end of file position now, so we would do the needful using `fseek()` so as to point the next location from EOF, & then begin appending.

At last we are reading the contents of file using `fgetc()` & printing contents on console.

## Categories of Files

File can be divided in two following categories

1. Text Mode
2. Binary Mode

## 'C' Scope

The difference between the two can be based on the following characteristics.

<b>Characteristics</b>	<b>Text Mode</b>	<b>Binary Mode</b>
New Line	In 'C' line ending is denoted by Newline character i.e. \n and its ASCII value is 10.	In Binary end of line is denoted by two different characters carriage return and linefeed i.e. \r and \n
End of File	In Text Mode a character 1A (Hexadecimal) indicates the EOF.	In binary It is just any number

Ex. 5.

Program to copy a file in to another file (source.c to target.c file)

```
#include<stdio.h>
void main()
{
    char ch;
    FILE *fs,*ft;
    clrscr();
    fs =fopen("source.c","r");
    ft =fopen("target.c","w");
    if(!fs)
    {
        printf("sorry file not found");
        exit();
    }
    while(1)
```

## 'C' Scope

```
{
  ch =getc(fs);
  putc(ch,ft);
  if (ch==EOF)
    break;
}
fclose(fs);
fclose(ft);
getch();
}
```

##### OUTPUT #####

In source file the matter is

This is a book

That is published by scope computer

after execution of this program the output will also be available in the target file

This is a book

That is published by scope computer

Ex. 6.

Program to copy source.c file into target.c in reverse order

```
#include<stdio.h>
void main()
{
  int m;
  char ch;
  FILE *fs,*ft;
  clrscr();
  fs =fopen("source.c","r");
  ft= fopen("target.c","w");
```

## 'C' Scope

---

```
if(!fs)
{
printf("Sorry file not found .... ");
exit();
}
m=-1;
do
{
fseek(fs,m,2);
ch=getc(fs);
if (ch=='\n')
m--;
putc(ch,ft);
m--;
}while(!fseek(fs,0,1));
fclose(fs);
fclose(ft);
getch();
}
```

Run part of above program:=

In source file we have written following matter

This is a book

That is published by scope computer

After running the above program we see the output in target.c file in following manner

### TARGET.C

retupmoc epocs yb dehsilbup si taht

koob a si siht\_

Ex. 8.



## 'C' Scope

Reverse the string without reversing the word

```
#include<stdio.h>
void main()
{
    int m=-1,i=0;
    char ch,str[30];
    FILE *fs,*ft;
    clrscr();
    fs =fopen("source.c","r");
    ft= fopen("target.c","w");
    if(!fs)
    {
        printf("sorry file not found");
        exit();
    }
    while( !fseek(fs,0,1) )
    {
        fseek(fs,m,2);
        ch =getc(fs);
        if(ch==' ' || ch ==EOF || ch== '\n')
        {
            for (i--;i>=0;i--)
                putc(str[i],ft);
            putc(ch,ft);
            if(ch=='\n')
                m--;
            i=0;
        }
        else
        {
            str[i]=ch;
            i++;
        }
    }
}
```

## 'C' Scope

```
}
m--;
}
for(i-=2;i>=0;i--)
    putc(str[i],ft);
fclose(fs);
fclose(ft);
getch();
}
```

##### OUTPUT #####

In source file we have written following matter

This is a book  
That is published by scope computer

After execution of the above program we see the output in target.c file in following manner

### TARGET.C

computer scope by published is that book a is this

```
#include<stdio.h>
void main()
{
    int i=0;
    char ch,str[30],temp,hold;
    FILE *fs,*ft;
    clrscr();
    fs =fopen("source.c", "r");
    ft= fopen("target.c","w");
    if(!fs)
```

## 'C' Scope

```
{
printf("sorry file not found");
exit();
}
hold = ' ';
while(1)
{
ch =getc(fs);
if(hold==' ' || hold=='\n')
temp=ch;
if(ch==' ' || ch ==EOF || ch == '\n' )
{
str[i]='\0';
fprintf(ft,str);
putc (temp,ft);
fprintf(ft,"a");
putc(ch,ft);
i=0;
}
else
{
if (hold!=' ' && hold!='\n')
{
str[i]=ch;
i++;
}
}
hold =ch;
if(ch ==EOF)
break;
}
fclose(fs);
fclose(ft);
```

## 'C' Scope

```
getch();
}
```

Lets see another program to fulfill the above mentioned task.

```
#include<stdio.h>
void main()
{
    int i=0;
    char ch,arr[30];
    FILE *fs,*ft;
    clrscr();
    fs =fopen("source.c","r");
    ft= fopen("target.c","w");
    if(!fs)
    {
        printf("sorry file not found");
        exit();
    }
    while(1)
    {
        ch =getc(fs);
        if(ch== '' || ch ==EOF || ch == '\n' )
        {
            for(i--;i>=0;i--)
                putc(arr[i],ft);
            putc(ch,ft);
            i=0;
            if(ch == EOF)
                break;
        }
        else
```

## 'C' Scope

```
{
    arr[i]=ch;
    i++;
}
}
fclose(fs);
fclose(ft);
getch();
}
```

Another way to deal with the same problem.

```
#include<stdio.h>
main()
{
FILE *fp,*ft;
char ch;
fp=fopen("book.c","r");
ft=fopen("book1.c","w");
while(ch!=EOF)
{
    ch=getc(fp);
    putc(ch,ft);
}
fclose(fp);
fclose(ft);
}
```

The above program progresses to copy the contents of one file into another. Till now we only did the conversation about file handling on the Turbo C++ IDE. Now we will discuss about Command Line arguments from the next section.

## 'C' Scope

We know a simple to print "hello world" can be made as given below

```
void main()
{
    printf("Hello World");
    getch();
}
Hello World
```

But in Turbo C++ IDE whenever we want to execute it we have to press Ctrl+F9 or run the snippet. If we wish to execute the program from outside the Turbo C++ IDE, & get rid of the hazzle of again & again pressing Ctrl+F9, we can use DOS prompt, just go at the directory where your program EXE file is kept & run using the file name, here my file name is ASSIGN103.C whose exe file is ASSIGN103.EXE

```
#include<stdio.h>

void main(int argc, char *argv[])
{
    printf("Hello World");
    getch();
}
```

For executing it form command prompt, we can write the following syntax, that contain .exe filename with no arguments

```
C:\TURBOC~1\Disk\TurboC3\SOURCE> ASSIGN103
```

## 'C' Scope

---

Hello

### Command Line Arguments

In environments that support C, there is a way to pass command-line arguments or parameters to a program when it begins executing. In C it is possible to accept command line arguments as user input for the program to be executed. Command-line arguments are given after the name of a program in command-line operating systems like DOS or Linux, and are passed in to the program from the operating system.

To use command line arguments in a program, we must first understand the full declaration of the main function, which previously has accepted no arguments. In fact, main can actually accept two arguments: one argument is number of command line arguments, and the other argument is a full list of all of the command line arguments.

The full declaration of main looks like this:

```
int main ( int argc, char *argv[ ] )
```

`argc` : the argument count that is the number of arguments passed into the program from the command line, including the name of the program.

`argv[ ]` : the argument vector, array of character pointers is the listing of all the arguments. Or we can say it is a pointer to an array of character strings that contain the arguments, one per string. By default, `argv[0]` is the name by which the program was invoked, so `argc` is at least 1.

## 'C' Scope

Lets see an example of this to practicaly understand the topic,

Ex. 8.

Program to implement use of command line arguments as input.

```
#include<stdio.h>
int main(int argc,char *argv[])
{
    int i;
    for(i=0;i<argc;i++)
    {
        printf("\n %s",argv[i]);
    }
    return 0;
}
```

#####OUTPUT#####

```
C:\TurboC++\Disk\TurboC3\SOURCE>jp_cmd      scope
computers point
```

```
C:\TURBOC~1\DISK\TURBOC3\SOURCE\JP_CMD.EX
E
scope
computers
point
```

The above program prints all the command line arguments with a new line feed. Above we showed a simple example of use of command line arguments, now we can code a



## 'C' Scope

program to make a copy of source file, as discussed in previous section ( using command line arguments ).

Ex. 9.

Program to copy contents of one file to another file, taking name of two files using command line arguments.

```
#include<stdio.h>
void main(int argc, char *argv[])
{
    char ch;
    FILE *fs,*ft;
    fs =fopen(argv[1],"r");
    ft= fopen(argv[2],"w");
    if(argc != 3)
    {
        printf("Sorry Invalid No. Of Argument:");
        printf("\n Usage  : file6 <SOURCE FILE NAME>
<TARGET FILE NAME>");
        exit();
    }
    if(!fs)
    {
        printf("sorry %s file not found",argv[1]);
        exit();
    }
    while(1)
    {
        ch = getc(fs);
        if(ch == EOF)
            break;
        else
            putc(ch,ft);
    }
}
```

## 'C' Scope

---

```
}  
fclose(fs);  
fclose(ft);  
}
```

### Self Review

- Q1. Write a program in 'C' that takes file name and mode from user and then perform any specific operation depending upon the mode?
- Q2. Write a program that takes command line arguments as data that is to be written in a text file?
- Q3. Write a program to save the output of one program of 'C' to a .txt file?
- Q4. Write a program to compare contents of two files, & tell which file has more contents according to ASCII ?
- Q5. Write a program to count the number of characters, words, lines in a file?
- Q6. Write a program to merge data of two files ,sort it & then write the sorted data in a new file?
- Q7. Write a program to implement database in file?
- Q8. Write a program to merge two & make a new merged data file that contains sorted data from two files?
- Q9. Write a program to find the size of a file?
- Q10. Can we do all work using getc() & putc()? State Yes or No? Give reasons in support of your answer?
- Q11. Explain the difference between various input syntices of files?

# CHAPTER - 11

## PREPROCESSOR DIRECTIVES

In 'C' programs you write the statement

```
#include <stdio.h>
```

why we write this statement ?

we will discuss this statement in three parts

1. #
2. include
3. <stdio.h>

here # is the preprocessor directive

include is the file inclusion directive

<stdio.h> is the header file.

The meaning of preprocessor directive is to process the attached statement before any type of processing.

Means to include the header file stdio.h file before any type of processing.

There are certain preprocessor directives

### **#include**

It directs the compiler to include given header file. It must be enclosed between the pair of angular brackets of double quotes.

## 'C' Scope

The #include directives can be used in three ways

1. #include <stdio.h>
2. #include "stdio.h"
3. #include "c:\turbo\c\stdio.h"

The first #include<stdio.h> indicates that the header file is in its default location.

The second #include "stdio.h" shows that the header file is available in current directory.

But if it is available in any other directory then you have to supply the path as given in point number 3.

## MACRO

Like function call we can also use macro call but there is a difference in function and macro calls.

In macro call the preprocessor blindly replace the macro template with its macro expansion.

### Simple MACRO

Consider the following program segment

```
#define MAX 10
#include <stdio.h>
main()
{
    int x;
    for (x=0;x<MAX;X++)
```

In this example MAX is called as macro template and its value can be called as macro expansion.

## 'C' Scope

```
printf (“%d\n”,x);  
}
```

Never use ; at the end of macro

in above example writing instead 10 in the loop we are using MAX which has already been defined in the first line of the program.

As we compile the program first it is checked by the preprocessor and where it finds the #define directive it checks the entire program to search macro template and wherever it finds the macro templates it simply replaced by appropriate macro expansion. It does not change macro written inside the string, such as

```
#define X 20  
printf (“The value of X is %d”,40);
```

X written inside the string will not get changed.

Macro template is generally written in CAPITAL letters (not necessarily) just to identify all the templates written in the program.

Some other forms of MACRO substitution

### MACRO with arguments

This is another form of macro which can perform more complex tasks and is more useful

The syntax of this form is

```
#define macro(argument list) expansion
```

## 'C' Scope

Now consider the following example

Ex. 1.

Program to demonstrate use of macro(s)

```
#define SQUARE(I) (I*I)
#include <stdio.h>
main()
{
    int a,b;
    a=10;
    b=SQUARE(a)
    printf ("The square of the given number is %d ",b);
    getch();
}
```

##### OUTPUT #####

The square of the given number is 100

In above program the macro would be expanded like

$B=(I * I)$

But consider the following program

Ex. 2.

Program to implement use argumented macros

```
#define SQUARE(i) (I*I)
#include <stdio.h>
main()
{
    int a,b,c;
    a=2;
    b=3
```

## 'C' Scope

```
b=SQUARE(a+b)
printf ("The square of the given number is %d ",c);
getch();
}
```

##### OUTPUT #####

The square of the given number is 11

Which is not correct

In above example macro would be expanded because it is blindly replaced by the template

```
C=SQUARE(a+b * a+B)
```

So, This error may be corrected as

```
#define SQUARE(i) ((i) * (i))
#include <stdio.h>
main()
{
    int a,b,c;
    clrscr();
    a=2;
    b=3;
    c=SQUARE(a+b);
    printf ("The square of the given number is %d ",c);
    getch();
}
```

##### OUTPUT #####

The square of the given number is 25

In this example it would be expanded as

```
SQUARE = ((a+b) * (a+b))
```

0

## 'C' Scope

Ex. 3.

Program to find whether character is in lower case or not using macro.

```
#include <stdio.h>
#define islow(chr) ((chr) >='a' && (chr) <='z') ? 1 : 0;
main()
{
    char ch;
    printf ("Enter a character : ");
    ch=getchar();
    if (islow(ch))
        printf ("It is lower case character : " );
    else
        printf ("It is not a lower case character : ");
    getch();
}
```

### Nesting Of Macros

It is possible to use a macro in the expansion of another macro this is called as macro nesting e.g.

```
#define SQUARE(x) ((x) * (x))
#define CUBE(x) (SQUARE(x) * (x))
#define SIXTH(x) (CUBE(x) * CUBE(x))
```

In above example preprocessor directive first expand

$((\text{SQUARE}(x) * (x)) * (\text{SQUARE}(x) * (x)))$   
and then finally it is further expanded into  
 $((((x) * (x)) * (x)) * ((x) * (x)))$



## 'C' Scope

i.e. x<sup>6</sup>

There are so many library functions can be used as macros and as true functions, these functions are defined in various header files.

Differences between the MACROS and functions

<b>S.No.</b>	<b>MACRO</b>	<b>Functions</b>
1.	A MACRO does not check the data types of argument passed. So it has less error checking facility. It only checks the number of argument	Arguments passed to a function are checked for their number and data types.
2.	We can not use a pointer in a MACRO.	A function can use the pointer.
3.	MACRO cannot call itself. I.e. recursion is not possible in MACRO	A function can call itself i.e. recursion is possible.
4.	When a MACRO is used it is simply replaced every time, this may result to unnecessarily increase the length of the program.	When a function is called a same function is used every time..

## 'C' Scope

### Examples of Invalid #define directives

**#define MAX 50;**

Invalid

(Semicolon is allowed at the end of MACRO)

**#define MAX=50**

Invalid

(= Assignment operator can not be used in a MACRO)

**# define MAX 50**

Invalid

(White space is not allowed in between # and the define statement)

**#define MAX 10 50**

Invalid

(More than two values can not be used in string)

**#define MAX-LEN 50**

Invalid

(- symbol is not allowed in MACRO)

**#define MAX 10, MIN 50**

Invalid

(More than two names can not be defined at a time)

The next directive that can be used in 'C' is #undef that is used to undefine the previously defined macro , defined using #define. Lets see an example:-

```
#define PI 3.14
void main()
{
    printf(“%f”,PI);
}
```

The output of above program will be :-

## 'C' Scope

3.14000

If we use #undef then

```
#define PI 3.14
#undef PI
void main()
{
    printf(“%f”,PI);
}
```

Then the code above will result an error that is undefined symbol PI used in line 5, in printf statement.

#ifdef & #ifndef

These two directives can be used to check whether the macro is defined previously or not, based on the result of that condition the compiler can perform some actions.

Lets see an example of this,

```
#define MIN_BALANCE 1000
#ifdef MIN_BALANCE
#undef MIN_BALANCE
#endif
#ifndef MIN_BALANCE
#define MIN_BALANCE 1500
#endif
void main()
{
    printf(“%d”,MIN_BALANCE);
}
```

## 'C' Scope

```
}
```

The above code shows that if MIN\_BALANCE is previously defined then undefined it & redefine it with value 1500 , Now the output of the program will be 1500.

We can also write the above code using **#ifdef... #else... #endif** construct

```
#define MIN_BALANCE 1000
#ifdef MIN_BALANCE
    #define MIN_BALANCE 1500
#else
    #define MIN_BALANCE 1000
#endif
```

```
void main()
{
    printf("%d",MIN_BALANCE);
}
```

In the above code if MIN\_BALANCE is defined previously then set its value to 1500 else set it to 1000 only.

We can also use else if construct using **#elif** as a preprocessor directive.

There are also many other preprocessor directives like:

- 1) **#line** :- The **#line** directive tells the preprocessor to change the compiler's internally stored line number

## 'C' Scope

and filename to a given line number and filename.  
Its syntax can be given as,

#line number "filename"

- 2) #error :- The #error directive tells the processor to abort the compilation process when it is found, generating a compilation the error that can be specified as its parameter.

#error token\_string

- 3) #pragma :- The #pragma directives offer a way for each compiler to offer machine- and operating system-specific features while retaining overall compatibility with the C.

#pragma token-string

Some of the pragma token\_strings can be : message, warning, alloc\_text, once, pack

## Summary

- The meaning of preprocessor directive is to process the attached statement before any type of processing.
- The include directive directs the compiler to include given header file. It must be enclosed between the pair of angular brackets of double quotes.
- The define directive is used to define a macro, in macro call the preprocessor blindly replace the macro template with its macro expansion.
- We can also nest some macros within another macro.

# Chapter 12

## STRUCTURES

Till now we have studied much about arrays and it feels that it is enough to store data of different types. But then why structure?

What is its use?

To answer these questions let us consider a situation where we have to store data of say a 100 number of employees in a company and we have to save following data of an employee for an employer:- name, age and salary. We see that these 3 attributes of the single entity employee are of different data types and as per the knowledge we have, after studying arrays, we can store all these data as following :-

```
char name[20];
```

This is the declaration to declare array "name" for maximum 19 width. In this array we can store the employees name.

```
int age;
```

This indicates that we have a variable age to store the age of the employee.

```
float salary;
```

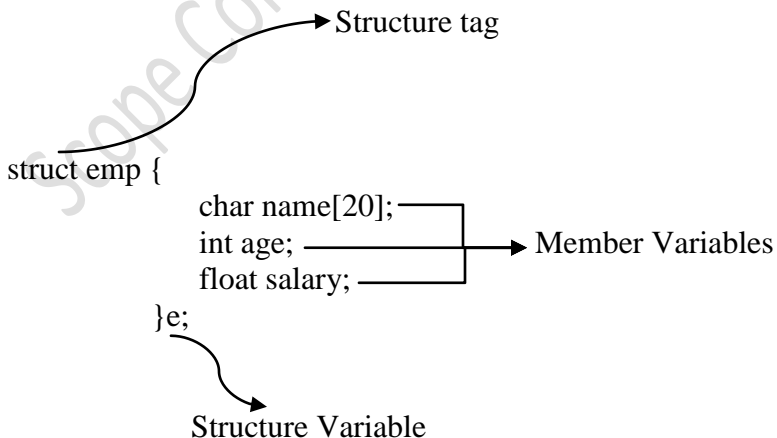
## 'C' Scope

This indicates that we have variable salary to store the salary of the employee.

But then these declarations will store the data of one employee in an independent way and it will not be connected with each other to show that it's a detail about one employee.

As soon as we enter more names, age and salary to our records we will be ending up getting a collection of several names, many age and different salaries but in no manners they will be connected to each other to show which age belongs to which employee or which salary belongs to a particular employee.

So we see that in real world different types of data are connected to each other and then only those data become meaningful. So, we have to combine all these fields in a packet such that it shows an association with each other and for that we have to use structure as follows :-



## 'C' Scope

Now with the use of structures the 3 attributes name, age and salary that belongs to one employee (e)

If we want to store data of more than one employee we can use array of structures and for that the declaration will be as follows :-

Suppose if we want to maintain the records for the students fees system. Some fields might be like this sname, enrollno, totalfees, class etc. for given situation structure will be declared as

```
struct stud {  
    char sname[20];  
    int enrollno;  
    int totalfees;  
    int class;  
}s;
```

```
struct emp  
{  
    int empno;  
    int age;  
    int salary;  
}e,e1,e2,e3;
```

Like the example given above we can name as many variables as we want, type structure, But the problem is acquainted where we have large number of records to be processed, lets say if we have 1000 employees in our company so according to the example above the last emp



## 'C' Scope

type variable will be e999, starting right from e,e1...e999. Also processing of these records will be troublesome. So for this purpose we can use Array of Structures

```
struct emp {  
    char name[20];  
    int age;  
    float salary;  
}e[10];
```

Structure tag

Structure Variable type array

Never compare two structure variables

We can also use more than one structure variable to refer same structure template

```
#include <stdio.h>  
main()  
{  
    struct emp  
    {  
        int empno;  
        int age;  
        int salary;  
    }e,e1,e2,e3;
```

The following example shows that that we assign the values of one structure variable to another structure

## 'C' Scope

variable of same template. In this we have a structure we have a structure definition and 2 structure variables e and e1 and we are assigning the values of e to e1 which is successfully works as shown in the example.

```
#include <stdio.h>
main()
{
    struct emp
    {
        int empno;
        int age;
        int salary;
    }e,e1;
    clrscr();
    e.empno=201;
    e.age=30;
    e.salary=50;
    e1=e;

    printf ("\n%d",e1.empno);
    printf ("\n%d",e1.age);
    printf ("\n%d",e1.salary);
}
```

The output would be

```
201
30
50
```

You cannot compare two structure variables which shown in the next example

## 'C' Scope

```
#include <stdio.h>
main()
{
    struct emp
        {
            int empno;
            int age;
            int salary;
        }e,e1;
    clrscr();
    e.empno=201;
    e.age=30;
    e.salary=50;
    e1=e;
    if(e1==e)
        printf ("values are equal");
    else
        printf ("Values are not equal");
}
```

This will produce an error because we are trying to compare two structure variable e and e1 which is not possible. Therefore, comparison of two structure variable is an illegal operation.

```
#include <stdio.h>
main()
{
    struct emp
        {
            int empno;
            int age;
```

## 'C' Scope

```
        int salary;
    };
    Struct emp e = {230,30,4000};
```

we can assign values directly at the time of declaration using the pair of curly braces.

Size of structure

```
#include <stdio.h>
main()
{
    struct emp
    {
        int empno;
        int age;
        int salary;
    }e;
    printf(size of (e));
}
```

In the above example the size of the structure is 6 bytes .

## Array of structures

```
#include <stdio.h>
main()
{
    struct emp
    {
        int empno;
        int age;
        int salary;
    }e;
```

## 'C' Scope

---

/\*To print a Mark sheet Using Structure \*/

```
#include <stdio.h>
main()
{
    struct stud {
        char name[30];
        char fname[30];
        int rno;
        int c,cpp,unix;
    }s[3];

    int i,total,per;
    clrscr();
    for (i=0;i<3;i++)
    {
        printf ("Enter the Student's Name :");
        gets(s[i].name);
```

Scope Computers 8560000535

## 'C' Scope

```
printf ("Enter the Father's Name :");
gets(s[i].fname);
printf ("Enter Roll Number : ");
scanf ("%d",&s[i].rno);
printf ("Enter the 'C' Language Marks :");
scanf ("%d",&s[i].c);
printf ("Enter the 'C++' Language Marks : ");
scanf ("%d",&s[i].cpp);
printf ("Enter the Unix Marks :");
scanf ("%d",&s[i].unix);
fflush(stdin);
}

clrscr();
for (i=0;i<3;i++)
{
printf ("\nName of Student : %s          Father's Name :
%s",s[i].name,s[i].f
printf ("\n-----");
printf ("\nSubject          Max.Marks      Min.Marks
Obtained :");
printf ("\n-----");
printf ("\nC      Language          100          33
%d",s[i].c);
printf ("\nC++ Language          100          33
%d",s[i].cpp);
printf ("\nUnix
%d",s[i].unix);
printf ("\n-----");
total=s[i].c+s[i].cpp+s[i].unix;
per=total/3;
printf ("\nTotal          300          99          %d",total);
printf ("\n-----");
```

## 'C' Scope

```
printf ("\n-----");
printf ("\n Percentage : %d      Division = ",per);
if (per >=60)
    printf ("First ");
else if (per >=45)
    printf("Second ");
else if (per >=33)
    printf ("Third :");
else
    printf ("Failed :");
printf ("\n\n Press Any Key to Continue ..... :");
getch();
clrscr();
}
}
```

##### OUTPUT #####

Enter the Student's Name :NARENDRA BHARTI

Enter the Father's Name :SHAILESH BHARTI

Enter Roll Number : 1250

Enter the 'C' Language Marks :67

Enter the 'C++' Language Marks : 70

Enter the Unix Marks :89

Enter the Student's Name :RANJAN BHATI

Enter the Father's Name :GUNJAN BHATI

Enter Roll Number : 1251

Enter the 'C' Language Marks :56

Enter the 'C++' Language Marks : 48

Enter the Unix Marks :57

Enter the Student's Name :ATUL SHARMA

Enter the Father's Name :PRAFULL SHARMA

## 'C' Scope

Enter Roll Number : 1252

Enter the 'C' Language Marks :45

Enter the 'C++' Language Marks : 36

Enter the Unix Marks :42

Name of Student : NARENDRA BHARTI

Father's

Name : SHAILESH BHARTI

---

Subject	Max.Marks	Min.Marks	Obtained :
C Language	100	33	67
C++ Language	100	33	70
Unix	100	33	89
	300	99	226

---

Percentage : 75      Division = First

Press Any Key to Continue ..... :

Name of Student : RANJAN BHATI

Father's Name :

GUNJAN BHATI

---

Subject	Max.Marks	Min.Marks	Obtained :
C Language	100	33	56
C++ Language	100	33	48
Unix	100	33	57
	300	99	161

---

Percentage : 53      Division = Second



## 'C' Scope

Press Any Key to Continue .....

Name of Student : ATUL SHARMA      Father's Name :  
PRAFULL SHARMA

---

Subject	Max.Marks	Min.Marks	Obtained :
C Language	100	33	45
C++ Language	100	33	36
Unix	100	33	42
	300	99	123

---

Percentage : 41      Division = Third :

Press Any Key to Continue .....

### Summary

- Structure is only a template based on which we can have a record like schema to put detail in.
- Always place the semi colon at the end of structure definition.
- Structure is like a
- We can use also use more than one structure variable to refer same structure template.

## **'C' Scope**

### **Self Review**

- Q1. What is the difference between structure, union & bit field? Explain.
- Q2. Write a program to make employee database file using structure & also make it dynamically available?
- Q3. Write a program to maintain sales billing & inventory using structures & files?
- Q4. Write the various input & output statements used for structures with files?
- Q5. Write a program to implement pointer to structures ?
- Q6. Write a program to implement structures using functions?

## 'C' Scope

### Operator's Precedence Table

Position	Operator	Name	Associativity
1	() []	Function Array	Left to Right
2	+ - ++ -- ! ~ * & sizeof (type)	Unary Plus Unary Minus Increment Decrement Not One's Complement Pointer Address of Size Type Casting	Right to Left
3.	* / %	Multiply Divide Modulus	Left to Right
4.	+ -	Add Subtract	Left to Right
5	<< >>	Left Shift Right Shift	
6.	< <= > >=	Less than Less than or Equal to Greater Than Greater than or Equal to	Left to Right
7.	== !=	Comparison Not Equal to	Left to Right

## 'C' Scope

8.	&	Bitwise AND	Left to Right
9	^	Bitwise XOR	Left to Right
10		Bitwise OR	Left to Right
11	&&	Logical AND	Left to Right
12		Logical OR	Left to Right
13.	? :	Ternary Operator	Left to Right
14.	= *= /= %= += -= &= ^=  = <<= >>=	Assignment	Right to Left
15.	,	Comma	Left to Right

**'C' Scope**

---

---

**Project**

Scope Computers 8560000535