

**‘C’  
Programming  
Language**

**SECTION – 1**

**LECTURE : 1-3**

'C' introduction, Tokens

---

**SECTION – 2**

**LECTURE : 4-5**

Control Statements ,Loop

---

**SECTION – 3**

**LECTURE : 6-7**

Array ,String

---

**SECTION – 4**

**LECTURE : 8-9**

Pointers

---

**SECTION – 5**

**LECTURE : 10**

Functions

---

**SECTION – 6**

**LECTURE : 11**

Structure, Union

---

**SECTION – 7**

**LECTURE : 12**

File Handling

---

\*\*\*\*\*

## SECTION - 1

\*\*\*\*\*

### 'C' introduction, Data Type, Operators

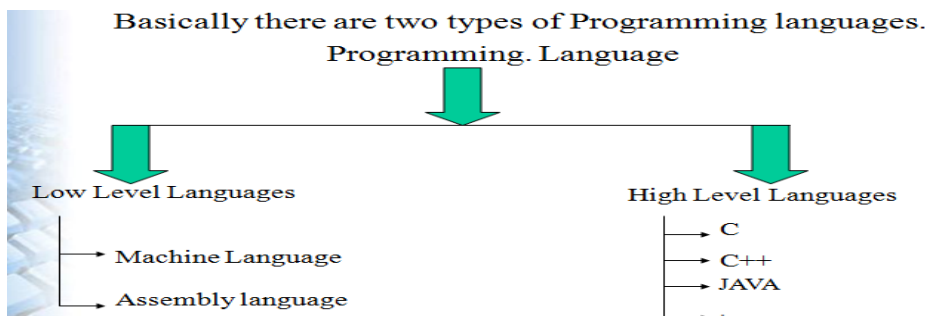
#### What is Language :

- Language is medium of communication.
- If two persons want to communicate with each other , they have to use a common language.
- Like Hindi, English, Punjabi etc.

#### What is Computer Programming:

- In the same way when a user wants to communicate with a computer ,the language that he/she uses for this purpose is known as programming language .Ex: C, C++, JAVA .....
- Computer programming is a way of giving computers instructions about what they should do next. These instructions are known as code, and computer programmers write code to solve problems or perform a task.

#### Types of programming Language:



#### Machine Language:

- It is in 0 and 1 format
- No need to translate.
- Programs are less understandable
- Difficult to write large programs
- Debugging is most difficult.
- Programs are machine dependent
- Not portable: portable to machine of the same architecture only

#### Assembly Language:

- Uses mnemonic for programming (ADD,STORE)
- Need assembler to translate from assembly to machine language
- Programs are more understandable then machine language and less understandable then High level language.
- Difficult to write large programs but easy then machine language
- Debugging is easy then machine language and complex then high level language
- Programs are machine dependent
- Not portable: portable to processor of the same architectures only

#### High Level Language:

- It uses natural language elements, which is easy to use.
- Need translator(compiler or assembler) to translate from high level language to object code
- Easy to understand
- Easy to write large programs and suitable for software development
- Debugging is easier than others

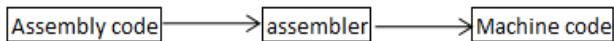
- Programs are not machine dependent
- Programs are portable

### Types of Translator:

1. Assembler
2. Compiler
3. Interpreter

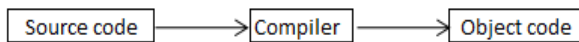
### Assembler:

- Assembly and machine language are low level language
- The assembly codes are easier to write but system can execute the machine code.
- So we need a translator (Assembler) to translate assembly code to machine code.



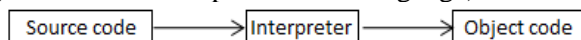
### Compiler:

- Used to translate high level source code to object code.
- Compiler scans all the lines of source program and list out all syntax errors at a time.
- It takes less time to execute.
- Object produced by compiler gets saved in a file. So file do not need to compile again and again



### Interpreter:

- Used to translate source code to object code.
- Interpreter scans one line at a time of source program if there is any syntax error , the execution of program terminates immediately.
- It takes more time to execute.
- Machine code produced by interpreter is not saved in any file . So we need to interpret the file each time(BASIC is an interpreter based language)



## 'C' Programming Tokens:

1. Keywords
2. Identifiers
3. Variables
4. Constants
5. Operators
6. Special Symbols

### 1. 'C' KEYWORDS:

- Keywords are the words whose meaning has already been explained to the C compiler
- The keywords **cannot** be used as variable names because if we do so we are trying to assign a new meaning to the keyword, which is not allowed by the computer.
- There are 32 keywords available in C.
- Rules to be followed for all programs written in C:
  - All keywords are in lower case.
  - C is case sensitive so **int** is different from **Int**.
  - The keywords **cannot** be used as variable names because if we do so we are trying to assign a new meaning to the keyword, which is not allowed by the computer.
- List of C programming keywords.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

## 2. IDENTIFIERS :

- Identifiers refer to the naming of programming elements like variables, functions and arrays.
- These are user-defined names and consist of sequence of letters and digits, with a letter as a first character.
- Both uppercase and lowercase letters are permitted, although lowercase letters are commonly used.
- The underscore character is also permitted in identifiers. It is usually used as a link between two words in long identifiers.
- Some legal identifier :

**arena, s\_count**  
**marks40**  
**class\_one**

- Some illegal identifiers:
  - **1stsst**
  - **oh!god**
  - **start....end**
- The number of characters in the variable that are recognized differs from compiler to compiler.
- An identifier cannot be the same as a C keyword.

## 3. VARIABLE :

- Variables are named locations in memory that are used to hold a value that may be modified by the program.
- Unlike constants that remain unchanged during the execution of a program.
- A variable may take different values at different times during execution.
- The syntax for declaring a variable is –

DataType IdentifierName ;

**Example.** int num;  
long int sum , a;

## 4. CONSTANT :

- Constants are named locations in memory that are used to hold a value that cannot be modified by the program.
- Constants remain unchanged during the execution of a program.
- **const keyword** is used to declare constant data.
- The syntax for declaring a constant is –

**const** DataType IdentifierName ;

**Example.** const float PI = 3.14f;

## 5. OPERATORS :

- C supports rich set of operators.
- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- Operators are used in programs to manipulate data and variables.
- Types of operators

### 1. **Unary operator**(+,-,++,--)

### 2. **Binary operator**

- Arithmetic operators (+,-,\*,/,%)
- Relational operators (<,>,<=,>=,==,!=)
- Logical operators(&&(logical AND),||(logical OR),!(logical Not))
- Assignment operators (=)
- Bitwise operators (~,>>,<<,&,&|)
  - C supports bitwise operators for manipulation of data at bit level.
  - Bitwise operators may not be applied to float or double.
  - Bitwise operators are as follows:
    - & bitwise AND
    - | bitwise OR
    - ^ bitwise exclusive OR
    - << shift left
    - >> shift right
    - ~ One's Complements
- Special operators (. ,sizeof(),&,->)

### 3. **Ternary operator**

- C supports a ternary operator i.e. conditional operator.
- The conditional operator is defined as

“ ? : ”

- The conditional operator works in following format:

Syntax : **(condition) ? true statement : false statement ;**

**for example:**

```
void main()
{
    int a=11, b=20;
    ( a==b) ? printf("both r Equal") : printf("Not Equal") ;
}
```

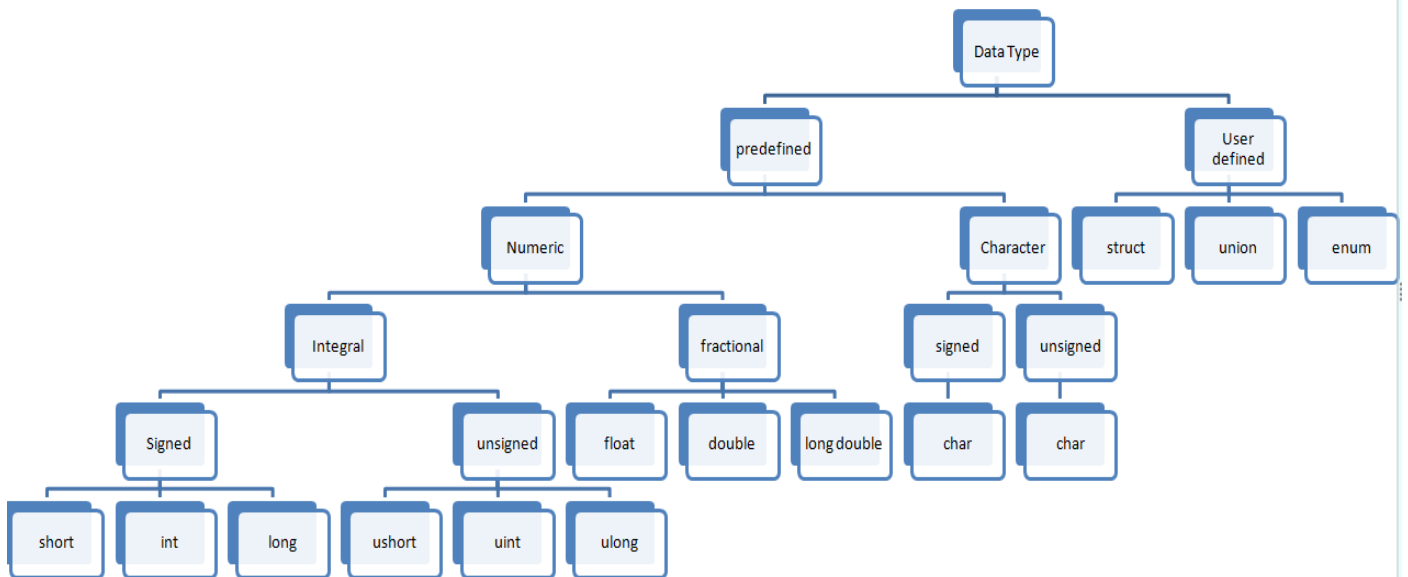
Output: Not Equal

## 6. SPECIAL OPERATOR :

- !, @, #, \$, &, \*, .....
- These all symbols are called Special Symbols.
- Every symbol has its special meaning in different respect at different place that's why it is called Special Symbols.

## Data Type:

- C language is rich in its data type.
- Data type tells the type of data, that you are going to store in memory.
- It gives the information to compiler that how much memory ( No. of bytes ) will be stored by the data.



Type	Size	Description
<b>char</b>	1 byte	Used for characters or integer variables.
<b>int</b>	2 or 4 bytes	Used for integer values.
<b>float</b>	4 bytes	Single precision floating point values
<b>double</b>	8 bytes	Double precision floating point values

In addition to these data types, some of them may be used with a modifier that affects the characteristics of the data object. These modifiers are listed in Table.

Modifier	Description
<b>long</b>	Forces a type int to be 4 bytes (32 bits) long and forces a type double to be larger than a double (but the actual size is implementation defined). Cannot be used with short.
<b>short</b>	Forces a type int to be 2 bytes (16 bits) long. Cannot be used with long.
<b>unsigned</b>	Causes the compiler (and CPU) to treat the number as containing only positive values. Because a 16-bit signed integer can hold values between – 32,768 and 32,767, an unsigned integer can hold values between 0 and 65,535. The unsigned modifier can be used with char, long, and short (integer) types.

## Storage Class:

To fully define a variable we need to mention :

1. Type of variable i.e ( int, char, float )
2. The storage class of a variable

The storage class is associated with a variable which decide what would be the default value of variable, where it would get stored, what will be the life of variable, and what will be the scope of variable.

Moreover, a variable's storage class tells us:

- Where the variable would be stored.
- What will be the initial value of the variable, if initial value is not specifically assigned.(i.e. the default initial value).
- What is the visibility of the variable; i.e. in which functions the value of the variable would be available.
- What is the scope or life of the variable; i.e. how long would the variable exist.

### **Types of Storage class :**

1. Automatic storage class
2. Register storage class
3. Static storage class
4. Extern storage class

#### **1. Automatic storage class : (this is default class)**

Keyword : auto  
Default value : garbage  
Storage : memory  
Scope of variable : Local to block  
Life of variable : till control remains in the block in which it is declared.

**Ex: auto int x;**

**Ex:**

```
#include<stdio.h> //header file
void main() //main function
{
auto int x=10; //variable declaration and initialization
{
auto int x=20;
printf(“%d”,x); //printf() method to print value of x
}
printf(“%d”,x);
}
Output :20 ,10
```

#### **2. Register storage class :**

Keyword : register  
Default value : garbage  
Storage : registers of CPU



Scope of variable : Local to block  
Life of variable : till control remains in the block in which it is declared.

**Ex: register int x;**

Ex:

```
Void main()
{
register int r;
for(r=1;r<=10;r++)
    Printf(“%d”,r);
}
```

Output:1,2,3,4,5,6,7,8,9,10

Note : If any CPU register is not available to store data then ‘c’ compiler automatically convert the storage class from register to automatic, because number of CPU registers are limited..

### 3. Static storage class

Keyword : static  
Default value : zero  
Storage : memory  
Scope of variable : local to block  
Life of variable : till the program is running, and it persist in various function call

**Ex: static int x;**

Ex:

```
void main()
{
static int x;
printf(“%d”,x);
x++;
if(x<=10)
main();
}
```

**Output:0,1,2,3,4,5,6,7,8,9,10**

```
Ex: extern int x;
int x=10;
void main ()
{
extern int x;
printf (“%d”, x);
}
```

**Output: 10|**

### 4. Extern storage class

Keyword : extern  
Default value : zero  
Storage : memory  
Scope of variable : global from point of declaration onwards  
Life of variable : till program execution does not come to an end

## MCQ

1. Which of the following is not a valid variable name declaration?

- a) int \_\_a3;
- b) int \_\_3a;
- c) int \_\_A3;

d) None of the mentioned

**Ans- d**

2. Which of the following is not a valid variable name declaration?

- a) int \_a3;
- b) int a\_3;
- c) int 3\_a;
- d) int \_3a

**Ans- c**

3. All keywords in C are in \_\_\_\_\_

- a) LowerCase letters
- b) UpperCase letters
- c) CamelCase letters
- d) None of the mentioned

**Ans- a**

4. Which of the following is true for variable names in C?

- a) They can contain alphanumeric characters as well as special characters
- b) It is not an error to declare a variable to be one of the keywords (like goto, static)
- c) Variable names cannot start with a digit
- d) Variable can be of any length

**Ans- c**

5. Which is valid C expression?

- a) int my\_num = 100,000;
- b) int my\_num = 100000;
- c) int my num = 1000;
- d) int \$my\_num = 10000;

**Ans- b**

6. int main()

```
{  
    int main = 3;  
    printf("%d", main);  
    return 0;  
}
```

- a) It will cause a compile-time error
- b) It will cause a run-time error
- c) It will run without any error and prints 3
- d) It will experience infinite looping

**Ans- c**

7. The format identifier '%i' is also used for \_\_\_\_\_ data type.

- a) char
- b) int
- c) float
- d) double

**Ans- b**

8. What is the size of an int data type?

- a) 4 Bytes
- b) 8 Bytes
- c) Depends on the system/compiler
- d) Cannot be determined

**Ans- c**

9. What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    float f1 = 0.1;
    if (f1 == 0.1)
        printf("equal\n");
    else
        printf("not equal\n");
}
```

- a) equal
- b) not equal
- c) output depends on the compiler
- d) error

**Ans- b**

10. What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    float x = 'a';
    printf("%f", x);
    return 0;
}
```

- a) a
- b) run time error
- c) compile time error
- d) 97.000000

**Ans- d**

11. What is the difference between the following 2 codes?

```
//Program 1
int main()
{
    int d, a = 1, b = 2;
    d = a++ + ++b;
```

```

    printf("%d %d %d", d, a, b);
}
//Program 2
int main()
{
    int d, a = 1, b = 2;
    d = a++ ++b;
    printf("%d %d %d", d, a, b);
}

```

- a) No difference as space doesn't make any difference, values of a, b, d are same in both the case  
b) Space does make a difference, values of a, b, d are different  
c) Program 1 has syntax error, program 2 is not  
d) Program 2 has syntax error, program 1 is not
12. What will be the output of the following C code?

**Ans- d**

```

#include <stdio.h>
int main()
{
    int a = 1, b = 1, c;
    c = a++ + b;
    printf("%d, %d", a, b);
}

```

- a) a = 1, b = 1  
b) a = 2, b = 1  
c) a = 1, b = 2  
d) a = 2, b = 2

**Ans- b**

13. What will be the output of the following C code?

```

#include <stdio.h>
int main()
{
    int a = 1, b = 1, d = 1;
    printf("%d, %d, %d", ++a + ++a+a++, a++ + ++b, ++d + d++ + a++);
}

```

- a) 15, 4, 5  
b) 9, 6, 9  
c) 9, 3, 5  
d) Undefined (Compiler Dependent)

**Ans- d**

14. For which of the following, "PI++;" code will fail?

- a) #define PI 3.14  
b) char \*PI = "A";  
c) float PI = 3.14;  
d) none of the Mentioned

**Ans- a**

15. What will be the output of the following C code?

```
int main()
{
    int a = 10, b = 10;
    if (a = 5)
        b--;
    printf("%d, %d", a, b--);
}
```

- a) a = 10, b = 9
- b) a = 10, b = 8
- c) a = 5, b = 9
- d) a = 5, b = 8

**Ans- c**

16. What will be the output of the following C code?

```
#include <stdio.h>
void main()
{
    int k = 8;
    int x = 0 == 1 && k++;
    printf("%d%d\n", x, k);
}
```

- a) 0 9
- b) 0 8
- c) 1 8
- d) 1 9

**Ans- b**

17. What will be the output of the following C code?

```
#include <stdio.h>
void main()
{
    char a = 'a';
    int x = (a % 10)++;
    printf("%d\n", x);
}
```

- a) 6
- b) Junk value
- c) Compile time error
- d) 7

**Ans- c**

18. What will be the output of the following C code snippet?

```
#include <stdio.h>
void main()
{
    1 < 2 ? return 1: return 2;
}
```

- a) returns 1
- b) returns 2
- c) Varies
- d) Compile time error

**Ans- d**

19. What will be the output of the following C code snippet?

```
#include <stdio.h>
void main()
{
    unsigned int x = -5;
    printf("%d", x);
}
```

- a) Run time error
- b) Aries
- c) -5
- d) 5

**Ans- c**

20. What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int x = 2, y = 1;
    x *= x + y;
    printf("%d\n", x);
    return 0;
}
```

- a) 5
- b) 6
- c) Undefined behaviour
- d) Compile time error

**Ans- b**

21. What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int x = 2, y = 0;
    int z = (y++) ? y == 1 && x : 0;
    printf("%d\n", z);
    return 0;
}
```

- a) 0
- b) 1
- c) Undefined behaviour
- d) Compile time error

**Ans- a**

22. What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int x = 1;
    int y = x == 1 ? getchar(): 2;
    printf("%d\n", y);
}
```

- a) Compile time error
- b) Whatever character getchar function returns
- c) Ascii value of character getchar function returns
- d) 2

**Ans- c**

23. What will be the output of the following C code?

```
int main()
{
    int x = 1;
    short int i = 2;
    float f = 3;
    if (sizeof((x == 2) ? f : i) == sizeof(float))
        printf("float\n");
    else if (sizeof((x == 2) ? f : i) == sizeof(short int))
        printf("short int\n");
}
```

- a) float
- b) short int
- c) Undefined behaviour
- d) Compile time error

**Ans- a**

24. What will be the output of the following C code?

```
int main()
{
    int a = 2;
    int b = 0;
    int y = (b == 0) ? a :(a > b) ? (b = 1): a;
    printf("%d\n", y);
}
```

- a) Compile time error
- b) 1
- c) 2
- d) Undefined behaviour

**Ans- c**

25. What will be the output of the following C code?

```
int main()
{
    int y = 1, x = 0;
    int l = (y++, x++) ? y : x;
```

```
printf("%d\n", 1);
}
```

- a) 1
- b) 2
- c) Compile time error
- d) Undefined behaviour

**Ans- a**

**Programs:**

- Q1. WAP to swap the value of two variables without using third variable.
- Q2. WAP to take a character value from the keyboard and print ascii value of this character.
- Q3. WAP to calculate reverse of a given number.(number of 4 digit )

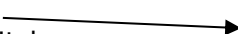
\*\*\*\*\*

**SECTION - 2**

\*\*\*\*\*

**Control Statement**

- C language has decision making capabilities and supports controlling of statements.
- C supports following control or decision making statements:

- |  |   |   |
|--|---|---|
| <ul style="list-style-type: none"> <li>1. IF</li> <li>2. Switch</li> <li>3. Break</li> <li>4. Continue</li> <li>5. Goto</li> </ul> |  | <p><b>Different forms of If statement</b></p> |
|--|---|---|

- 1. Simple If statement.
- 2. If.....else statement.
- 3. Nested if.....else statement.
- 4. Else if ladder or else if ...else .

**Syntax of if else=>**

```
if(Condition)
{
    True statement...1
    True statement...2
    ...
}
else
{
    False statement 1
    False statement 1
    ...
}
```

**Switch – case statement**

- The control statement that allows us to make a decision from the number of choices is called a **switch**
- The switch-case control statement is a multi-way decision maker that tests the value of an expression against a list of integers or character constants.
- When a match is found, the statements associated with that constant are executed.
- a **switch-case-default**, since these three keywords go together to make up the control statement. They most often appear as follows:

```
switch ( integer/character choice )
{
    case constant 1 :
```



```

        do this ;
case constant 2 :
    do this ;

default :
    do this ;
}

```

### **Break statement**

We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the conditional test. The keyword **break** allows us to do this.

Syntax: For( ; ; )

```

{ -----
  break;
  -----
}

```

### **continue Statement**

In some programming situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed. The keyword **continue** allows us to do this. When **continue** is encountered inside any loop, control automatically passes to the beginning of the loop.

Syntax: For( ; ; )

```

{ -----
  break;
  -----
}

```

### **Goto statement**

- C supports the go to statement to branch unconditionally from one point to another in the program.

- It requires a label in order to identify the place where branch is to be made.

syntax is:

```

goto label;
Statement 1;
Statement 2;
-----
label:

```

## **Loops**

The versatility of the computer lies in its ability to perform a set of instructions repeatedly. This involves repeating some portion of the program either a specified number of times or until a particular condition is being satisfied. This repetitive operation is done through a loop control instruction.

There are three types of loop

3. While
4. For
5. Do...while

### **Difference between while and do...while**

While	Do...while
<ol style="list-style-type: none"> <li>1. While loop is entry control loop</li> <li>2. In While loop the condition is tested first if the condition is true then the statements are executed</li> <li>3. while loop do not run in case the condition given is false</li> <li>4. while loop do not run in case the condition given is false</li> <li>5. In a while loop the condition is first tested and if it returns true then it goes in the loop</li> <li>6. Syntax: <b>while(condition)</b>  <pre> {     statement 1;     statement 2;..... } </pre> </li> </ol>	<ol style="list-style-type: none"> <li>1. do while is exit control loop.</li> <li>2. In do while the statements are executed for the first time and then the conditions are tested, if the condition turns out to be true then the statements are executed again.</li> <li>3. A do while loop runs at least once even though the condition given is false</li> <li>4. A do while is used for a block of code that must be executed at least once.</li> <li>5. A do while is used for a block of code that must be executed at least once.</li> <li>6. Syntax: <b>do</b>  <pre> {     statement 1;     statement 2;..... } while(condition); </pre> </li> </ol>

## MCQ - Control Statements

1. What will be the output of the following C code?

```
#include <stdio.h>
void main()
{
    int x = 5;
    if (x < 1)
        printf("hello");
    if (x == 5)
        printf("hi");
    else
        printf("no");
}
```

- a) hi
- b) hello
- c) no
- d) error

**Ans- a**

2. What will be the output of the following C code?

```
int x;
void main()
{
    if (x)
        printf("hi");
    else
        printf("how are u");
}
```

- a) hi
- b) how are you
- c) compile time error
- d) error

**Ans- b**

```
3. void main()
{
    int x = 5;
    if (true);
        printf("hello");
}
```

- a) hello
- b) error
- c) Nothing will be displayed
- d) Compiler dependent

**Ans- a**

```
4. void main()
{
    int x = 0;
    if (x == 0)
        printf("hi");
    else
        printf("how are u");
        printf("hello");
}
```

- a) hi
- b) how are you
- c) hello
- d) hihello

**Ans- d**

```
5. void main()
{
    int x = 5;
    if (x < 1);
        printf("Hello");

}
```

- a) Nothing
- b) Run time error
- c) Hello
- d) Varies

**Ans- c**

6. What will be the output of the following C code? (Assuming that we have entered the value 1 in the standard input)

```
#include <stdio.h>
void main()
{
    double ch;
    printf("enter a value between 1 to 2:");
    scanf("%lf", &ch);
    switch (ch)
    {
        case 1:
            printf("1");
            break;
        case 2:
            printf("2");
            break;
    }
}
```

- a) Compile time error
- b) 1
- c) 2
- d) Varies

**Ans- a**

7. What will be the output of the following C code? (Assuming that we have entered the value 1 in the standard input)

```
#include <stdio.h>
void main()
{
    char *ch;
    printf("enter a value between 1 to 3:");
    scanf("%s", ch);
    switch (ch)
    {
        case "1":
            printf("1");
            break;
        case "2":
            printf("2");
            break;
    }
}
```

- a) 1

- b) 2
- c) Compile time error
- d) No Compile time error

**Ans- c**

8. What will be the output of the following C code? (Assuming that we have entered the value 1 in the standard input)

```
#include <stdio.h>
void main()
{
    int ch;
    printf("enter a value between 1 to 2:");
    scanf("%d", &ch);
    switch (ch)
    {
        case 1:
            printf("1\n");
        default:
            printf("2\n");
    }
}
```

- a) 1
- b) 2
- c) 1 2
- d) Run time error

**Ans- c**

9. What will be the output of the following C code? (Assuming that we have entered the value 2 in the standard input)

```
void main()
{
    int ch;
    printf("enter a value between 1 to 2:");
    scanf("%d", &ch);
    switch (ch)
    {
        case 1:
            printf("1\n");
            break;
            printf("Hi");
        default:
            printf("2\n");
    }
}
```

- a) 1
- b) Hi 2
- c) Run time error

d) 2

**Ans- d**

10. What will be the output of the following C code? (Assuming that we have entered the value 2 in the standard input)

```
void main()
{
    int ch;
    printf("enter a value between 1 to 2:");
    scanf("%d", &ch);
    switch (ch)
    {
        printf("Hi");
        case 1:
            printf("1\n");
            break;
        default:
            printf("2\n");
    }
}
```

a) 1

b) Hi 2

c) Run time error

d. 2

**Ans- d**

0. Which keyword can be used for coming out of recursion?

a) break

b) return

c) exit

d) both break and return

**Ans- b**

11. What will be the output of the following C code?

```
int main()
{
    int a = 0, i = 0, b;
    for (i = 0; i < 5; i++)
    {
        a++;
        continue;
    }
}
```

a) 2

b) 3

c) 4

d) 5

**Ans- d**

```
12. int main()
{
    int a = 0, i = 0, b;
    for (i = 0; i < 5; i++)
    {
        a++;
        if (i == 3)
            break;
    }
}
```

- a) 1
- b) 2
- c) 3
- d) 4

**Ans- d**

13. The keyword 'break' cannot be simply used within \_\_\_\_\_

- a) do-while
- b) if-else
- c) for
- d) while

**Ans- b**

14. Which keyword is used to come out of a loop only for that iteration?

- a) break
- b) continue
- c) return
- d) none of the mentioned

**Ans- b**

15. What will be the output of the following C code?

```
void main()
{
    int i = 0, j = 0;
    for (i = 0; i < 5; i++)
    {
        for (j = 0; j < 4; j++)
        {
            if (i > 1)
                break;
        }
        printf("Hi \n");
    }
}
```

- a) Hi is printed 5 times
- b) Hi is printed 9 times

- c) Hi is printed 7 times
- d) Hi is printed 4 times

**Ans- a**

16. What will be the output of the following C code?

```
void main()
{
    int i = 0;
    int j = 0;
    for (i = 0; i < 5; i++)
    {
        for (j = 0; j < 4; j++)
        {
            if (i > 1)
                continue;
            printf("Hi \n");
        }
    }
}
```

- a) Hi is printed 9 times
- b) Hi is printed 8 times
- c) Hi is printed 7 times
- d) Hi is printed 6 times

**Ans- b**

17. What will be the output of the following C code?

```
void main()
{
    int i = 0;
    for (i = 0; i < 5; i++)
        if (i < 4)
        {
            printf("Hello");
            break;
        }
}
```

- a) Hello is printed 5 times
- b) Hello is printed 4 times
- c) Hello
- d) Hello is printed 3 times

**Ans- c**

18. What will be the output of the following C code?

```
void main()
{
    int i = 0;
    if (i == 0)
```



```
{
    printf("Hello");
    continue;
}
}
```

- a) Hello is printed infinite times
- b) Hello
- c) Varies
- d) Compile time error

**Ans- d**

19. What will be the output of the following C code?

```
void main()
{
    int i = 0;
    if (i == 0)
    {
        printf("Hello");
        break;
    }
}
```

- a) Hello is printed infinite times
- b) Hello
- c) Varies
- d) Compile time error

**Ans- d**

20. What will be the output of the following C code?

```
int main()
{
    int i = 0;
    do
    {
        i++;
        if (i == 2)
            continue;
        printf("In while loop ");
    } while (i < 2);
    printf("%d\n", i);
}
```

- a) In while loop 2
- b) In while loop in while loop 3
- c) In while loop 3
- d) Infinite loop

**Ans- a**

21. int main()

```

{
    int i = 0;
    while (i < 2)
    {
        if (i == 1)
            break;
        i++;
        if (i == 1)
            continue;
        printf("In while loop\n");
    }
    printf("After loop\n");
}

```

a)

In while loop

After loop

b) After loop

c)

In while loop

In while loop

After loop

d) In while loop

**Ans- b**

22. int main()

```

{
    int i = 0;
    char c = 'a';
    while (i < 2)
    {
        i++;
        switch (c)
        {
            case 'a':
                printf("%c ", c);
                break;
                break;
        }
    }
    printf("after loop\n");
}

```

a) a after loop

b) a a after loop

c) after loop

d) error

**Ans- b**

```
23.int main()
{
    printf("before continue ");
    continue;
    printf("after continue\n");
}
```

- a) Before continue after continue
- b) Before continue
- c) After continue
- d) Compile time error

**Ans- d**

```
24.int main()
{
    printf("%d ", 1);
    goto l1;
    printf("%d ", 2);
l1:goto l2;
    printf("%d ", 3);
l2:printf("%d ", 4);
}
```

- a) 1 4
- b) Compilation error
- c) 1 2 4
- d) 1 3 4

**Ans- a**

```
25. int main()
{
    printf("%d ", 1);
    goto l1;
    printf("%d ", 2);
}
void foo()
{
    l1 :
    printf("3 ", 3);
}
```

- a) 1 2 3
- b) 1 3
- c) 1 3 2
- d) Compilation error

**Ans- d**

**Programs:**

**Q1.** Write a program to find the greatest number among three, and print all numbers in ascending order.

**Q2.** WAP to find and display the product of three positive integer values based on the rule mentioned below:

It should display the product of the three values except when one of the integer value is 7. In that case, 7 should not be included in the product and the values to its left also should not be included. If there is only one value to be considered, display that value itself. If no values can be included in the product, display -1.

**Note:** Assume that if 7 is one of the positive integer values, then it will occur only once. Refer the sample I/O given below.

Sample Input	Expected Output
153>=1, 5, 3	15
378>=3, 7, 8	8
743>=7, 4, 3	12
157>=1, 5, 7	-1

**Q3.** WAP to generate and display the next date of a given date. Assume that

- Date is provided as day, month and year as shown in below table.
- The input provided is always valid. Output should be day-month-year.

**Hint:** print(day,"-",month,"-",year) will display day-month-year

Sample Input	Expected Output
Day 1	2-9-2015
Month 9	
Year 2015	

**Q4.** Food-Corner home delivers vegetarian and non-vegetarian combos to its customer based on order.

A vegetarian combo costs Rs.120 per plate and a non-vegetarian combo costs Rs.150 per plate. Their non-veg combo is really famous that they get more orders for their non-vegetarian combo than the vegetarian combo.

Apart from the cost per plate of food, customers are also charged for home delivery based on the distance in kms from the restaurant to the delivery point. The delivery charges are as mentioned below:

Distance in kms	Delivery charge in Rs per km
For first 3kms	0
For next 3kms	3
For the remaining	6

Given the type of food, quantity (no. of plates) and the distance in kms from the restaurant to the delivery point, write a python program to calculate the final bill amount to be paid by a customer.

The below information must be used to check the validity of the data provided by the customer:

- Type of food must be 'V' for vegetarian and 'N' for non-vegetarian.
- Distance in kms must be greater than 0.
- Quantity ordered should be minimum 1.

If any of the input is invalid, the bill amount should be considered as -1.

**Q5.** WAP to calculate  $a^b$  (power (a, b)).

**Q6.** WAP to calculate factors of any given number, and print addition of all these factors.

**Q7.** Write a general purpose function to convert any given year into its roman equivalent. The following table shows the roman equivalent of decimal numbers:

DECIMAL	ROMAN	DECIMAL	ROMAN
1	I	100	c
5	v	500	d
10	x	1000	m
50	l		

Example:

Roman equivalent of 1988 is mdccccl xxxviii

Roman equivalent of 1525 is mdxxv

**Q8.** Write a program to generate all combinations of 1,2 and 3 using for loop.

**Q9.** Write an application that calculates the squares and cubes of the numbers from 0 to 10 and prints the resulting values in table format, as follows

Number	Square	Cube
0	0	0
1	1	1
2	2	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

\*\*\*\*\*

## SECTION - 3

\*\*\*\*\*

### ARRAYS IN C

So far we have used only the fundamentals data types, namely char, int, float, double and variations of int and double. Although these types are very useful, they are constrained by the fact that a variable of these types can store only one value at any given time. Therefore they can be used to handle limited amounts of data. In many applications, however, we need to handle a large volume of data in terms of reading, processing and printing.

To process such large amounts of data, we need a powerful datatype that would facilitate efficient storing, accessing and manipulation of data items. C supports a derived data type known as "Array" that can be used for such applications.

In definition, an array is a fixed-size sequenced collection of elements of the same data type. In its simplest form, they can be used to represent a list of numbers or a number of names. These lists shares only one name or a common name. For instance, we can use an array name salary to represent a set of salaries of a group of employees in an organization. We can refer to the individual salary by writing a number called index number of subscript in bracket after the array name. For example:

salary[9]

represents the salary of 10th employee, as indices are started from 0. While the complete set of values is referred to as an array, individual values are called elements.

These powerful abilities enable us to develop concise and efficient programs. For example, we can use a loop construct, with the subscript as the control variable to read entire array, perform calculations and print out the results.

We will discuss the following types of arrays:

1. One-Dimensional Arrays
2. Two-Dimensional Arrays
3. Multi-Dimensional Arrays

### How to declare an array?

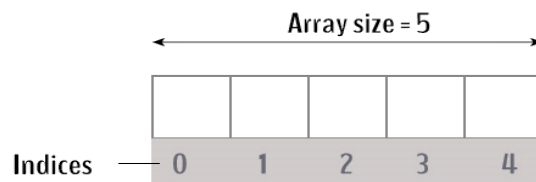
```
dataType arrayName[arraySize];
```

**For example,**

```
float mark[5];
```

Here, we declared an array, mark, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values.

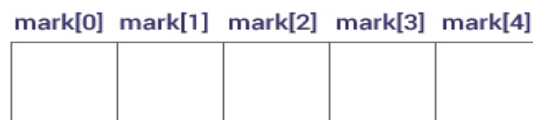
It's important to note that the size and type of an array cannot be changed once it is declared.



### Access Array Elements

You can access elements of an array by indices.

Suppose you declared an array mark as above. The first element is mark[0], the second element is mark[1] and so on.



### Few keynotes:

Arrays have 0 as the first index, not 1. In this example, mark[0] is the first element.

If the size of an array is n, to access the last element, the n-1 index is used. In this example, mark[4]

Suppose the starting address of mark[0] is **2120d**. Then, the address of the mark[1] will be **2124d**. Similarly, the address of mark[2] will be **2128d** and so on.

This is because the size of a float is 4 bytes.

### How to initialize an array

It is possible to initialize an array during declaration. For example,

- `int mark[5] = {19, 10, 8, 17, 9};`

You can also initialize an array like this.

- `int mark[] = {19, 10, 8, 17, 9};`

Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

### Input and Output Array Elements

Here's how you can take input from the user and store it in an array element.

```
// take input and store it in the 3rd element
scanf("%d", &mark[2]);
```

```
// take input and store it in the ith element
```

```
scanf("%d", &mark[i-1]);
```

Here's how you can print an individual element of an array.

- `// print the first element of the array`
- `printf("%d", mark[0]);`
  
- `// print the third element of the array`
- `printf("%d", mark[2]);`
  
- `// print ith element of the array`
- `printf("%d", mark[i-1]);`

### Example: Array Input/Output

- `// Program to take 5 values from the user and store them in an array`
- `// Print the elements stored in the array`
- `#include <stdio.h>`
- `int main() {`
- `int values[5];`
- 
- `printf("Enter 5 integers: ");`
- 
- `// taking input and storing it in an array`
- `for(int i = 0; i < 5; ++i) {`
- `scanf("%d", &values[i]);`
- `}`
- `printf("Displaying integers: ");`
- 
- `// printing elements of an array`
- `for(int i = 0; i < 5; ++i) {`
- `printf("%d\n", values[i]);`
- `}`
- `return 0;`
- `}`

### Output

```
Enter 5 integers: 1
-3
34
0
3
Displaying integers: 1
-3
34
0
3
```

In C programming, you can create an array of arrays. These arrays are known as multidimensional arrays. For example,

- `float x[3][4];`

Here, x is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Similarly, you can declare a three-dimensional (3d) array. For example,

- `float y[2][4][3];`

Here, the array y can hold 24 elements.

### Initializing a multidimensional array

Here is how you can initialize two-dimensional and three-dimensional arrays:

```
// Different ways to initialize two-dimensional array
```

```
int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};
```

```
int c[][3] = {{1, 3, 0}, {-1, 5, 9}};
```

```
int c[2][3] = {1, 3, 0, -1, 5, 9};
```

You can initialize a three-dimensional array in a similar way like a two-dimensional array. Here's an example,

```
int test[2][3][4] = {
  {{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},
  {{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};
```

### Example: Two-dimensional array to store and print values

```

• // C program to store temperature of two cities of a week and display it.
• #include <stdio.h>
• const int CITY = 2;
• const int WEEK = 7;
• int main()
• {
•   int temperature[CITY][WEEK];
•
•   // Using nested loop to store values in a 2d array
•   for (int i = 0; i < CITY; ++i)
•   {
•     for (int j = 0; j < WEEK; ++j)
•     {
•       printf("City %d, Day %d: ", i + 1, j + 1);
•       scanf("%d", &temperature[i][j]);
•     }
•   }
•   printf("\nDisplaying values: \n\n");
•
•   // Using nested loop to display vlues of a 2d array
•   for (int i = 0; i < CITY; ++i)
•   {
•     for (int j = 0; j < WEEK; ++j)
•     {
•       printf("City %d, Day %d = %d\n", i + 1, j + 1, temperature[i][j]);
•     }
•   }
•   return 0; }
•
•

```



## MCQ – ARRAYS IN C

1) What is an Array in C language.?

- A) A group of elements of same data type.
- B) An array contains more than one element
- C) Array elements are stored in memory in continuous or contiguous locations.
- D) All the above.

**Answer: D**

2) Choose a correct statement about C language arrays.

- A) An array address is the address of first element of array itself.
- B) An array size must be declared if not initialized immediately.
- C) Array size is the sum of sizes of all elements of the array.
- D) All the above

**Answer: D**

3) What are the Types of Arrays.?

- A) int, long, float, double
- B) struct, enum
- C) char
- D) All the above

**Answer: D**

4) An array Index starts with.?

- A) -1
- B) 0
- C) 1
- D) 2

**Answer: B**

5) Choose a correct statement about C language arrays.

- A) An array size can not changed once it is created.
- B) Array element value can be changed any number of times
- C) To access Nth element of an array students, use students[n-1] as the starting index is 0.
- D) All the above

**Answer: D**

6) What is the output of C Program.?  
`int main() { int a[]; a[4] = {1,2,3,4}; printf("%d", a[0]); }`

- A) 1
- B) 2
- C) 4
- D) Compiler error

**Answer: D**

Explanation: If you do not initialize an array, you must mention ARRAY SIZE.

7) What is the output of C Program.?

```
int main(){
    int a[];
    a[4] = {1,2,3,4};
    printf("%d", a[0]); }
```

- A) 1,5
- B) 2,6
- C) 0 0
- D) Compiler error

**Answer: A**

Explanation: It is perfectly allowed to skip array size if you are initializing at the same time. a[0] is first element. `int a[] = {1,2,3,4};`

8) What is the output of C Program?

```
int main() {
    char grade[] = {'A','B','C'};
    printf("GRADE=%c, ", *grade);
    printf("GRADE=%d", grade); }
```

- A) GRADE=some address of array, GRADE=A
- B) GRADE=A, GRADE=some address of array
- C) GRADE=A, GRADE=A
- D) Compiler error

**Answer: B**

Explanation: Variable grade = address of first element. \*grade is the first element of array i.e grade[0].

9) What is the output of C program?

```
int main()
{
    char grade[] = {'A','B','C'};
    printf("GRADE=%d, ", *grade);
    printf("GRADE=%d", grade[0]); }
```

- A) A A
- B) 65 A
- C) 65 65
- D) None of the above

**Answer: C**

Explanation: \*grade == grade[0]. We are printing with %d not with %c. So, ASCII value is printed.

10) What is the output of C program.?

```
int main()
{
    float marks[3] = {90.5, 92.5, 96.5};
    int a=0;
```

```

while(a<3)
{
    printf("%.2f", marks[a]);
    a++;
}

```

- A) 90.5 92.5 96.5      B) 90.50 92.50 96.50      C) 0.00 0.00 0.00      D) Compiler error

**Answer: B**

Explanation: 0.2%f prints only two decimal points. It is allowed to use float values with arrays.

**11) What is the output of C Program.?**

```

int main()
{
    int a[3] = {10,12,14};
    a[1]=20;
    int i=0;
    while(i<3)
    {
        printf("%d ", a[i]);
        i++; }
}

```

- A) 20 12 14      B) 10 20 14      C) 10 12 20      D) Compiler error

**Answer: B**

Explanation: a[i] is (i+1) element. So a[1] changes the second element.

**12) What is the output of C program?**

```

int main()
{
    int a[3] = {10,12,14};
    int i=0;
    while(i<3)
    {
        printf("%d ", i[a]);
        i++;
    }
}

```

- A) 14 12 10      B) 10 10 10      C) 10 12 14      D) None of the above

**Answer: C**

Explanation: a[k] == k[a]. Use any notation to refer to array elements.

**13) What is the output of C Program.?**

```

int main() {
    int a[3] = {20,30,40};
    a[0]++;
    int i=0;
    while(i<3)
    {
        printf("%d ", i[a]);
        i++;
    }
}

```

- A) 20 30 40      B) 41 30 20      C) 21 30 40      D) None of the above

**Answer: C**

Explanation: You can use increment and decrement operators on array variables too.

**14) What is the output of C program with arrays?**

```

int main() {
    int a[3] = {20,30,40};
    int b[3];
    b=a;
    printf("%d", b[0]); }

```

- A) 20      B) 30      C) address of 0th element.      D) Compiler error

**Answer: D**

Explanation: You can assign one array variable to other.

**15) What is the output of C Program with arrays and pointers.?**

```

int main() {

```

```
int a[3] = {20,30,40};
int (*p)[3];
p=&a;
printf("%d", (*p)[0]); }
```

- A) 20                      B) 0                      C) address of element 20                      D) Compiler error

**Answer: A**

Explanation: You cannot directly assign one array variable to other. But using an array pointer, you can point to the another array. (\*p) parentheses are very important.

**16) What is the output of C program with arrays and pointers?**

```
int main() {
    int a[3] = {20,30,40};
    int *p[3];
    p=&a;
    printf("%d", *p[0]); }
```

- A) 20                      B) address of element 20                      C) Garbage value                      D) Compiler error

**Answer: D**

Explanation: To point to an array, array pointer declaration should be like (\*p)[3] with parantheses. It points to array of 3 elements.

**17) What is the output of C program with arrays and pointers?**

```
int main() {
    int a[3] = {20,30,40};
    printf("%d", *(a+1)); }
```

- A) 20                      B) 30                      C) 40                      D) Compiler error

**Answer: B**

Explanation: \*(a+0) == \*a == a[0]. So \*(a+1) is element at index 1. Index starts with ZERO.

**18) What is an array Base Address in C language?**

- A) Base address is the address of 0th index element.  
 B) An array b[] base address is &b[0]  
 C) An array b[] base address can be printed with printf("%d", b);  
 D) All the above

**Answer: D**

**19) What is the output of C Program with arrays and pointers?**

```
void change(int[]);
int main() {
    int a[3] = {20,30,40};
    change(a);
    printf("%d %d", *a, a[0]); }
```

```
void change(int a[]) {
    a[0] = 10; }
```

- A) 20 20                      B) 10 20                      C) 10 10                      D) 20 30

**Answer: C**

Explanation: Notice that function change() is able to change the value of a[0] of main(). It uses Call By Reference. So changes in called function affected the original values.

**20) An entire array is always passed by \_\_\_ to a called function.**

- A) Call by value                      B) Call by reference  
 C) Address relocation                      D) Address restructure

**Answer: B**

**Programs:**

**Q1.** WAP to calculate binary of a given decimal number.

**Q2.** WAP to calculate hexadecimal of a given octal number.

**Q3.** WAP to remove duplicate elements from the array of 10 size.

**Q4.** WAP to count occurrence of each element from the array of size 10.

**Q5.** WAR to calculate average of array elements.

## Strings in C

Strings in C are actually arrays of characters. Although using pointers in C is an advanced subject, fully explained later on, we will use pointers to a character array to define simple strings, in the following manner:

```
char * name = "John Smith";
```

This method creates a string which we can only use for reading. If we wish to define a string which can be manipulated, we will need to define it as a local character array:

```
char name[] = "John Smith";
```

This notation is different because it allocates an array variable so we can manipulate it. The empty brackets notation `[]` tells the compiler to calculate the size of the array automatically. This is in fact the same as allocating it explicitly, adding one to the length of the string:

```
char name[] = "John Smith";  
/* is the same as */  
char name[11] = "John Smith";
```

The reason that we need to add one, although the string John Smith is exactly 10 characters long, is for the string termination: a special character (equal to 0) which indicates the end of the string. The end of the string is marked because the program does not know the length of the string - only the compiler knows it according to the code.

```
char c[] = "c string";
```

When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character `\0` at the end by default.

c		s	t	r	i	n	g	\0
---	--	---	---	---	---	---	---	----

### How to declare a string?

Here's how you can declare strings:

```
char s[5];
```

s[0]	s[1]	s[2]	s[3]	s[4]

Here, we have declared a string of 5 characters.

### How to initialize strings?

- You can initialize strings in a number of ways.
- `char c[] = "abcd";`
- 
- `char c[50] = "abcd";`
- 
- `char c[] = {'a', 'b', 'c', 'd', '\0'};`
- 
- `char c[5] = {'a', 'b', 'c', 'd', '\0'};`

c[0]	c[1]	c[2]	c[3]	c[4]
a	b	c	d	\0

Let's take another example:

```
char c[5] = "abcde";
```

Here, we are trying to assign 6 characters (the last character is `\0`) to a char array having 5 characters. This is bad and you should never do this.

### Read String from the user

You can use the `scanf()` function to read a string.

The `scanf()` function reads the sequence of characters until it encounters **whitespace** (space, newline, tab etc.).

### Example 1: `scanf()` to read a string

```
• #include <stdio.h>
• int main()
• {
•     char name[20];
•     printf("Enter name: ");
•     scanf("%s", name);
•     printf("Your name is %s.", name);
•     return 0;
• }
```

#### Output

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

Even though Dennis Ritchie was entered in the above program, only "Ritchie" was stored in the name string. It's because there was a space after Dennis.

### How to read a line of text?

You can use the `fgets()` function to read a line of string. And, you can use `puts()` to display the string.

### Example 2: `fgets()` and `puts()`

```
• #include <stdio.h>
• int main()
• {
•     char name[30];
•     printf("Enter name: ");
•     fgets(name, sizeof(name), stdin); // read string
•     printf("Name: ");
•     puts(name); // display string
•     return 0;
• }
```

#### Output

```
Enter name: Tom Hanks
Name: Tom Hanks
```

Here, we have used `fgets()` function to read a string from the user.

```
fgets(name, sizeof(name), stdin); // read string
```

The `sizeof(name)` results to 30. Hence, we can take a maximum of 30 characters as input which is the size of the name string.

To print the string, we have used `puts(name);`.

**Note:** The `gets()` function can also be to take input from the user. However, it is removed from the C standard.

It's because `gets()` allows you to input any length of characters. Hence, there might be a buffer overflow.

### Passing Strings to Functions

Strings can be passed to a function in a similar way as arrays. Learn more about passing arrays to a function.

### Example 3: Passing string to a Function

```
• #include <stdio.h>
• void displayString(char str[]);
•
• int main()
• {
•     char str[50];
•     printf("Enter string: ");
•     fgets(str, sizeof(str), stdin);
```

```

•   displayString(str); // Passing string to a function.
•   return 0;
•   }
•   void displayString(char str[])
•   {
•   printf("String Output: ");
•   puts(str);
•   }

```

## Strings and Pointers

Similar like arrays, string names are "decayed" to pointers. Hence, you can use pointers to manipulate elements of the string. We recommended you to check C Arrays and Pointers before you check this example.

### Example 4: Strings and Pointers

```

•   #include <stdio.h>
•
•   int main(void) {
•   char name[] = "Harry Potter";
•
•   printf("%c", *name); // Output: H
•   printf("%c", *(name+1)); // Output: a
•   printf("%c", *(name+7)); // Output: o
•
•   char *namePtr;
•
•   namePtr = name;
•   printf("%c", *namePtr); // Output: H
•   printf("%c", *(namePtr+1)); // Output: a
•   printf("%c", *(namePtr+7)); // Output: o
•   }

```

## Commonly Used String Functions

- **strlen()** - calculates the length of a string
- **strcpy()** - copies a string to another
- **strcmp()** - compares two strings
- **strcat()** - concatenates two strings

## Multiple Choice Questions: Strings in C

### 1) What is a String in C Language?

- A) String is a new Data Type in C
- B) String is an array of Characters with null character as the last element of array.
- C) String is an array of Characters with null character as the first element of array
- D) String is an array of Integers with 0 as the last element of array.

Answer: B

### 2) Choose a correct statement about C String.

- ```
char ary[]="Hello..!";
```
- A) Character array, ary is a string.
  - B) ary has no Null character at the end
  - C) String size is not mentioned
  - D) String can not contain special characters.

**Answer: A**

Explanation: It is a simple way of creating a C String. You can also define it like the below. \0 is mandatory in this version. `char ary[] = {'h','e','l','l','o','\0'};`

### 3) What is the Format specifier used to print a String or Character array in C Printf or Scanf function.?

- A) %c
- B) %C
- C) %s
- D) %w

**Answer: C**

Explanation: `char ary[]="Hello..!"; printf("%s",ary);`

### 4) What is the output of C Program with Strings?

```
int main() {
```

```

char ary[]="Discovery Channel";
printf("%s",ary);
return 0;
}

```

- A) D                      B) Discovery Channel                      C) Discovery                      D) Compiler error

**Answer: B**

Explanation: %s prints the whole character array in one go.

**5) What is the output of C Program with Strings?**

```

int main() {
char str[]={g,'l','o','b','e'};
printf("%s",str);
return 0;
}

```

- A) g                      B) globeC) globe\0                      D) None of the above

**Answer: D**

Explanation: Notice that you have not added the last character \0 in the char array. So it is not a string. It can not know the end of string. So it may print string with some garbage values at the end.

**6) What is the output of C Program with Strings?**

```

int main() {
char str[]={g,'l','o','b','y','\0'};
printf("%s",str);
return 0;
}

```

- A) g                      B) globeC) globe\0                      D) Compiler error

**Answer: B**

Explanation: Adding a NULL or \0 at the end is a correct way of representing a C string. You can simple use char str[]="globy". It is same as above.

**7) How do you convert this char array to string?**

```

char str[]={g,'l','o','b','y'};

```

- A) str[5] = 0;                      B) str[5] = '\0'                      C) str[]={g,'l','o','b','y','\0'};                      D) All the above

**Answer: D**

**8) What is the output of C Program?**

```

int main() {
int str[]={g,'l','o','b','y'};
printf("A%c ",str);
printf("A%s ",str);
printf("A%c ",str[0]);
return 0;
}

```

- A) A A A                      B) A Ag Ag  
C) A\*randomchar\* Ag Ag                      D) Compiler error

**Answer: C**

Explanation: Notice that STR is not a string as it is not a char array with null at the end. So STR is the address of array which is converted to Char by %c. If you use %s, it prints the first number converted to char.

**9) What is the output of C Program with arrays?**

```

int main() {
char str[]={ "C","A","T","\0"};
printf("%s",str);
return 0;
}

```

- A) C                      B) CAT                      C) CAT\0                      D) Compiler error

**Answer: D**

Explanation: Yes. You can not use Double Quotes " to represent a single character. Correct way is 'C' not "C". You should use Single Quotes around a single character constant.

**10) What is the maximum length of a C String?**

- A) 32 characters B) 64 characters C) 256 characters D) None of the above

**Answer: D**

Explanation: Maximum size of a C String is dependent on implemented PC memory. C does not restrict C array size or String Length.

**11) What is the output of C program with strings?**

```
int main() {
    char str1[]="JOHN";
    char str2[20];
    str2= str1;
    printf("%s",str2);
    return 0;
}
```

- A) JOHN B) J C) JOHN\0 D) Compiler error

**Answer: D**

Explanation: You can not assign one string to the other. It is an error. " error: assignment to expression with array type

**12) What is the output of C Program with arrays?**

```
int main() {
    char str[25];
    scanf("%s", str);
    printf("%s",str);
    return 0;
} //input: South Africa
```

- A) South B) South Africa C) S D) Compiler error

**Answer: A**

Explanation: SCANF can not accept a string with spaces or tabs. So SCANF takes only South into STR.

**13) What is the output of C program with strings?**

```
int main() {
    char str[2];
    scanf("%s", str);
    printf("%s",str);
    return 0;
} //Input: South
```

- A) So B) South C) Compiler error D) None of the above

**Answer: B**

Explanation: In C Arrays, Overflow or Out of Bounds is not checked properly. It is your responsibility to check.

**14) What is the output of C Program with strings?**

```
int main() {
    char str[2];
    int i=0;
    scanf("%s", str);
    while(str[i] != '\0')
    {
        printf("%c", str[i]);
        i++;
    }
    return 0;
} //Input: KLMN
```

- A) KL B) KLMN C) Compiler error D) None of the above

**Answer: B**

Explanation: It always overwrites the next memory locations of the array. It is your responsibility to check bounds. scanf automatically adds a '\0' at the end of entered string.

**15) What is the output of C Program with String Pointer?**

```
int main() {
    char country[]="BRAZIL";
    char *ptr;
```



```

ptr=country;
while(*ptr != '\0')
{
    printf("%c", *ptr);
    ptr++;
}
return 0;
}

```

- A) B                      B) BRAZIL                      C) Compiler error                      D) None of the above

**Answer: B**

Explanation: \*ptr != '\0' is the main part of traversing a C String.

**16) How do you accept a Multi Word Input in C Language?**

- A) SCANF                      B) GETS                      C) GETC                      D) FINDS

**Answer: B**

Explanation: Yes. gets(str) fills the array str with the input given by the user.

**17) Choose a correct C Statement about Strings.**

- A) PRINTF is capable of printing a multi word string.  
 B) PUTS is capable of printing a multi word string.  
 C) GETS is capable of accepting a multi word string from console or command prompt  
 D) All the above

**Answer: D**

**18) What is the output of C Program with String Pointers?**

```

int main() {
    char *p1 = "GOAT";
    char *p2;
    p2 = p1;
    printf("%s", p2);
}

```

- A) G                      B) GOAT                      C) Compiler error                      D) None of the above

**Answer: B**

Explanation: Yes. You can assign one String pointer to another. But you can not assign a normal character array variable to other like STR2 = STR1. It is an error.

**19) What is the output of C Program with String arrays?**

```

int main() {
    char *p1 = "GOAT";
    char *p2;
    p2 = p1;
    p2="ANT";
    printf("%s", p1);
}

```

- A) ANT                      B) GOAT                      C) G                      D) A

**Answer: B**

Explanation: \*p1 and \*p2 are completely pointing different memory locations. So, p1 value is not touched.

**20) What is the output of C Program with String Arrays?**

```

int main() {
    char p[] = "GODZILLA";
    int i=0;
    while(p[i] != '\0')
    {
        printf("%c",*(p+i));
        i++;
    }
}

```

- A) G                      B) GODZILLA                      C) Compiler error                      D) None of the above

**Answer: B**

Explanation: Notice the usage of \*(p+i). Remember that, p[i] == \*(p+i) == \*(i+p) == i[p]

**Programs:**

- Q1. Write a program to remove the characters which have odd index of a given string.
- Q2. Write a program to count repeated characters in a string.

Sample string: 'thequickbrownfoxjumpsoverthelazydog'

Expected output :

o 4  
e 3  
u 2  
h 2  
r 2  
t 2

- Q3. WAP to take a string from the keyboard and remove extra spaces from it.
- Q4. WAP to reverse a string without using string function.
- Q5. WAP to take 10 students name from the keyboard and print all names in ascending order

\*\*\*\*\*

**SECTION -4 \*\*\*\*\***

**Pointers in C :**

*A pointer is a variable whose value is the address of another variable ie. direct address of the memory location. Like any variable or constant, you must declare a pointer before you can use it to store any variable address. The general form of a pointer variable declaration is:*

```
type *var-name;
```

Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable. The asterisk \* you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration:

```
int *ip; /* pointer to an integer */
double *dp; /* pointer to a double */
float *fp; /* pointer to a float */
char *ch /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

### How to use Pointers?

There are few important operations which we will do with the help of pointers very frequently.

**(a)** we define a pointer variables

**(b)** assign the address of a variable to a pointer and

**(c)** finally access the value at the address available in the pointer variable.

This is done by using unary operator `*` that returns the value of the variable located at the address specified by its operand. Following example makes use of these operations:

```
#include <stdio.h>
void main ()
{
    int var = 20; /* actual variable declaration */
    int *ip; /* pointer variable declaration */

    ip = &var; /* store address of var in pointer variable*/

    printf("Address of var variable: %x\n", &var );

    /* address stored in pointer variable */
    printf("Address stored in ip variable: %x\n", ip);

    /* access the value using the pointer */
    printf("Value of *ip variable: %d\n", *ip);
}
```

When the above code is compiled and executed, it produces result something as follows:

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

### NULL Pointers in C

It is always a good practice to assign a NULL value to a pointer variable in case you do not have exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.

The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program:

```
#include <stdio.h>

int main ()
{
    int *ptr = NULL;

    printf("The value of ptr is : %x\n", ptr );

    return 0;
}
```

When the above code is compiled and executed, it produces following result:

```
The value of ptr is 0
```

### Dangling Pointer in C

**Dangling pointers** arise when an object is deleted or de-allocated, without modifying the value of the **pointer**, so that the **pointer** still points to the memory location of the de-allocated memory. In short **pointer** pointing to non-existing memory location is called **dangling pointer**

#### Pointer Arithmetic:

C pointer is an address which is a numeric value. Therefore, you can perform arithmetic operations on a pointer just as you can a numeric value. There are four arithmetic operators that can be used on pointers: ++, --, +, and -

##### ✓ Incrementing a Pointer

Incrementing a pointer, which increases its value by the number of bytes of its data type as shown below:

```
int x = 30;           //Suppose Address of X is 200
int *p = &x;         //Suppose Address of P is 100
p++;                 //Increment the address of P by 2 i.e. 202 because size of int is 2 bytes
```

##### ✓ Decrementing a Pointer

The same considerations apply to decrementing a pointer, which decreases its value by the number of bytes of its data type as shown below:

```
int x = 30;           //Suppose Address of X is 200
int *p = &x;         //Suppose Address of P is 100
p--;                 //Decrement the address of P by 2 i.e. 198 because size of int is 2 bytes
```

##### ✓ Pointer Comparisons

Pointers may be compared by using relational operators, such as ==, <, and >. If p1 and p2 point to variables that are related to each other, such as elements of the same array, then p1 and p2 can be meaningfully compared.

##### ✓ Following arithmetic operations are valid with pointers:-

```
p1 = p1 + 4;
```

```
p1 = p1 - 2;
```

```
p3 = p1 - p2;
```

##### ✓ Following other operations are valid with pointers:-

```
p1 > p2
```

```
p1 == p2
```

```
p1 != p2
```

```
p1++;
```

```
--p2;
```

##### ✓ Following operations are not valid with pointers:-

```
p3 = p1 + p2;
```

```
p3 = p1 / p2;
```

```
p3=p1*p2;
```

```
p1=p1/3;
```

### Using Pointers as function argument : Call by reference

The **call by reference** method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.

To pass the value by reference, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following function **swap()**, which exchanges the values of the two integer variables pointed to by its arguments.

```
/* function definition to swap the values */
void swap(int *x, int *y)
{
    int temp = *x; /* save the value at address x */
    *x = *y; /* put y into x */
    *y = temp; /* put temp into y */
}
```

```
#include <stdio.h>
void swap(int *x, int *y); // function prototype
void main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;
    printf("Before swap, value of a : %d\n", a );
    printf("Before swap, value of b : %d\n", b );
    swap(&a, &b); //calling swap by passing addresses of a and b
    printf("After swap, value of a : %d\n", a );
    printf("After swap, value of b : %d\n", b );
}
```

Let us put above code in a single C file, compile and execute it, it will produce following result:

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :200
After swap, value of b :100
```

### Difference between call by value & call by reference.

| Cal by Value                                                                                                 | Call by Reference                                                    |
|--------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
| This is the usual method to call a function in which only the value of the variable is passed as an argument | In this method, the address of the variable is passed as an argument |

|                                                                                                                         |                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Any alternation in the value of the argument passed is local to the function and is not accepted in the calling program | Any alternation in the value of the argument passed is accepted in the calling program(since alternation is made indirectly in the memory location using the pointer) |
| Memory location occupied by formal and actual arguments is different                                                    | Memory location occupied by formal and actual arguments is same and there is a saving of memory location                                                              |
| Since a new location is created, this method is slow                                                                    | Since the existing memory location is used through its address, this method is fast                                                                                   |

## Pointers in Detail

Pointers have many but easy concepts and they are very important to C programming. The following important pointer concepts should be clear to any C programmer –

| Sr.No. | Concept & Description                                                                                                                                                                       |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1      | <b>Pointer arithmetic</b><br><b>There are four arithmetic operators that can be used in pointers: ++, --, +, -</b>                                                                          |
| 2      | <b>Array of pointers</b><br><b>You can define arrays to hold a number of pointers.</b>                                                                                                      |
| 3      | <b>Pointer to pointer</b><br><b>C allows you to have pointer on a pointer and so on.</b>                                                                                                    |
| 4      | <b>Passing pointers to functions in C</b><br><b>Passing an argument by reference or by address enable the passed argument to be changed in the calling function by the called function.</b> |
| 5      | <b>Return pointer from functions in C</b><br><b>C allows a function to return a pointer to the local variable, static variable, and dynamically allocated memory as well.</b>               |

## MCQ – Pointers in C

1. What will be the output of the following C code?

```
int main()
{
    char *p = NULL;
    char *q = 0;
```

```

if (p)
    printf(" p ");
else
    printf("nullp");
if (q)
    printf("q\n");
else
    printf(" nullq\n");
}

```

- a) nullp nullq
- b) Depends on the compiler
- c) x nullq where x can be p or nullp depending on the value of NULL
- d) p q

**Ans: a**

2. What will be the output of the following C code?

```

int main()
{
    int i = 10;
    void *p = &i;
    printf("%d\n", (int)*p);
    return 0;
}

```

- a) Compile time error
- b) Segmentation fault/runtime crash
- c) 10
- d) Undefined behaviour

**Ans: a**

3. What will be the output of the following C code?

```

int main()
{
    int i = 10;
    void *p = &i;
    printf("%f\n", *(float*)p);
    return 0;
}

```

- a) Compile time error
- b) Undefined behaviour
- c) 10
- d) 0.000000

**Ans: d**

4. What will be the output of the following C code?

```

int *f();
int main()
{
    int *p = f();
    printf("%d\n", *p);
}
int *f()
{
    int *j = (int*)malloc(sizeof(int));
    *j = 10;
    return j;
}

```

- a) 10
- b) Compile time error
- c) Segmentation fault/runtime crash since pointer to local variable is returned

d) Undefined behaviour

**Answer: a**

5. What will be the output of the following C code?

```
int *f();
int main()
{
    int *p = f();
    printf("%d\n", *p);
}
int *f()
{
    int j = 10;
    return &j;
}
```

- a) 10
- b) Compile time error
- c) Segmentation fault/runtime crash
- d) Undefined behaviour

**Answer: a**

Explanation: We are returning address of a local variable which should not be done. In this specific instance, we are able to see the value of 10, which may not be the case if we call other functions before calling printf() in main().

6. Comment on the following pointer declaration.

```
int *ptr, p;
```

- a) ptr is a pointer to integer, p is not
- b) ptr and p, both are pointers to integer
- c) ptr is a pointer to integer, p may or may not be
- d) ptr and p both are not pointers to integer

**Answer: a**

7. What will be the output of the following C code?

```
int main()
{
    int *ptr, a = 10;
    ptr = &a;
    *ptr += 1;
    printf("%d,%d/n", *ptr, a);
}
```

- a) 10,10
- b) 10,11
- c) 11,10
- d) 11,11

**Answer: d**

8. Comment on the following C statement.

```
const int *ptr;
```

- a) You cannot change the value pointed by ptr
- b) You cannot change the pointer ptr itself
- c) You May or may not change the value pointed by ptr
- d) You can change the pointer as well as the value pointed by it

**Answer: a**

9. Which is an indirection operator among the following?

- a) &
- b) \*



- c) ->
- d) .

**Answer: b**

10. Which of the following does not initialize ptr to null (assuming variable declaration of a as int a=0;)?

- a) int \*ptr = &a;
- b) int \*ptr = &a - &a;
- c) int \*ptr = a - a;
- d) All of the mentioned

**Answer: a**

11. What will be the output of the following C code?

```
int x = 0;
void main()
{
    int *ptr = &x;
    printf("%p\n", ptr);
    x++;
    printf("%p\n ", ptr);
}
```

- a) Same address
- b) Different address
- c) Compile time error
- d) Varies

**Answer: a**

12. What will be the output of the following C code?

```
int x = 0;
void main()
{
    int *const ptr = &x;
    printf("%p\n", ptr);
    ptr++;
    printf("%p\n ", ptr);
}
```

- a) 0 1
- b) Compile time error
- c) 0xbf605e8 0xbf605ec
- d) 0xbf605e8 0xbf605e8

**Answer: b**

13. What will be the output of the following C code?

```
void main()
{
    int x = 0;
    int *ptr = &x;
    printf("%p\n", ptr);
    ptr++;
    printf("%p\n ", ptr);
}
```

- a) 0xbf605e8 0xbf605ec
- b) 0xbf605e8 0xbf60520
- c) 0xbf605e8 0xbf605e9
- d) Run time error

**Answer: a**

14. What will be the output of the following C code?

```
void main()
{
    int x = 0;
    int *ptr = &5;
    printf("%p\n", ptr);
}
```

- ```
}  
a) 5  
b) Address of 5  
c) Nothing  
d) Compile time error
```

**Answer: d**

15. What will be the output of the following C code?

```
void main()  
{  
    int x = 0;  
    int *ptr = &x;  
    printf("%d\n", *ptr);  
}
```

- a) Address of x  
b) Junk value  
c) 0  
d) Run time error

**Answer: c**

16. What will be the output of the following C code?

```
void main()  
{  
    int a[3] = {1, 2, 3};  
    int *p = a;  
    printf("%p\t%p", p, a);  
}
```

- a) Same address is printed  
b) Different address is printed  
c) Compile time error  
d) Nothing

**Answer: a**

17. What will be the output of the following C code?

```
void main()  
{  
    char *s = "hello";  
    char *p = s;  
    printf("%p\t%p", p, s);  
}
```

- a) Different address is printed  
b) Same address is printed  
c) Run time error  
d) Nothing

**Answer: b**

18. What will be the output of the following C code?

```
void main()  
{  
    char *s = "hello";  
    char *p = s;  
    printf("%c\t%c", p[0], s[1]);  
}
```

- a) Run time error  
b) h h  
c) h e  
d) h l

**Answer: c**

19. What will be the output of the following C code?

```
void main()
{
    char *s= "hello";
    char *p = s;
    printf("%c\t%c", *(p + 3), s[1]);
}
```

- a) h e
- b) ll
- c) l o
- d) l e

**Answer: d**

20. What will be the output of the following C code?

```
#include <stdio.h>
void main()
{
    char *s= "hello";
    char *p = s;
    printf("%c\t%c", 1[p], s[1]);
}
```

- a) h h
- b) Run time error
- c) ll
- d) e e

**Answer: d**

21. What will be the output of the following C code?

```
void foo( int[] );
int main()
{
    int ary[4] = {1, 2, 3, 4};
    foo(ary);
    printf("%d ", ary[0]);
}
void foo(int p[4])
{
    int i = 10;
    p = &i;
    printf("%d ", p[0]);
}
```

- a) 10 10
- b) Compile time error
- c) 10 1
- d) Undefined behaviour

**Answer: c**

22. What will be the output of the following C code?

```
int main()
{
    int ary[4] = {1, 2, 3, 4};
    int *p = ary + 3;
    printf("%d\n", p[-2]);
}
```

- a) 1
- b) 2
- c) Compile time error
- d) Some garbage value

**Answer: b**

```

23. int main()
{
    int ary[4] = {1, 2, 3, 4};
    int *p = ary + 3;
    printf("%d %d\n", p[-2], ary[*p]);
}

```

- a) 2 3
- b) Compile time error
- c) 2 4
- d) 2 somegarbagevalue

**Answer: d**

24. What will be the output of the following C code?

```

int main()
{
    int ary[4] = {1, 2, 3, 4};
    printf("%d\n", *ary);
}

```

- a) 1
- b) Compile time error
- c) Some garbage value
- d) Undefined variable

**Answer: a**

25. What will be the output of the following C code?

```

int main()
{
    const int ary[4] = {1, 2, 3, 4};
    int *p;
    p = ary + 3;
    *p = 5;
    printf("%d\n", ary[3]);
}

```

- a) 4
- b) 5
- c) Compile time error
- d) 3

**Answer: b**

### Programs:

**Q1.**WAP to swap value of two int variable through pointer.

**Q2.** WAP to Accept two int value without using '&' sign in scanf() and display sum of variable.

**Q3.** WAP to Search any value in Array without using index number.

**Q4.** Write a program to count repeated characters in a string using pointer.

Sample string: 'thequickbrownfoxjumpsoverthelazydog'

Expected output :

```

o 4
e 3
u 2
h 2

```

r 2  
t 2

Q5. WAP to take a string from the keyboard and remove all vowel characters using pointer.

\*\*\*\*\*

## SECTION - 5

\*\*\*\*\*

# Functions in C

In c, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by `{}`. A function can be called multiple times to provide reusability and modularity to the C program. In other words, we can say that the collection of functions creates a program. The function is also known as *procedure* or *subroutine* in other programming languages.

---

### Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.
- However, Function calling is always a overhead in a C program.

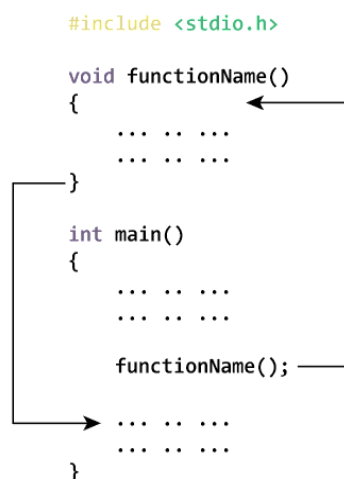
- **Function declaration** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.
- **Function call** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.
- **Function definition** It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.

SN	C function aspects	Syntax
1	Function declaration	return_type function_name (argument list);
2	Function call	function_name (argument_list)
3	Function definition	return_type function_name (argument list) {function body;}

The syntax of creating function in c language is given below:

- return\_type function\_name(data\_type parameter...){
- //code to be executed
- }

How function works in C programming?



## Types of Functions

There are two types of functions in C programming:

- **Library Functions:** are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.

- **User-defined functions:** are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.

#### Return Value

A C function may or may not return a value from the function. If you don't have to return any value from the function, use void for the return type.

Let's see a simple example of C function that doesn't return any value from the function.

#### **Example without return value:**

- **void** hello(){
- printf("hello c");
- }

If you want to return any value from the function, you need to use any data type such as int, long, char, etc. The return type depends on the value to be returned from the function.

Let's see a simple example of C function that returns int value from the function.

#### **Example with return value:**

- **int** get(){
- **return** 10;
- }

In the above example, we have to return 10 as a value, so the return type is int. If you want to return floating-point value (e.g., 10.2, 3.1, 54.5, etc), you need to use float as the return type of the method.

- **float** get(){
- **return** 10.2;
- }

Now, you need to call the function, to get the value of the function.

#### Different aspects of function calling

A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

- g. function without arguments and without return value
- h. function without arguments and with return value
- i. function with arguments and without return value
- j. function with arguments and with return value

#### Example for Function without argument and return value

##### **Example 1**

```
#include<stdio.h>
void printName();
void main ()
{
    printf("Hello ");
    printName();
}
void printName()
{
    printf("Javatpoint");
}
```

#### **Output**

Hello Javatpoint

## MCQ: Functions in C

1) Choose correct statement about Functions in C Language.

- A) A Function is a group of c statements which can be reused any number of times.
- B) Every Function has a return type.
- C) Every Function may no may not return a value.
- D) All the above.

**Answer: D**

2) Choose a correct statement about C Language Functions.

- A) A function name can not be same as a predefined C Keyword.
- B) A function name can start with an Underscore( \_ ) or A to Z or a to z.
- C) Default return type of any function is an Integer.
- D) All the above.

**Answer: D**

3) Choose a correct statement about C Function.?

```
main() {  
    printf("Hello");  
}
```

- A) "main" is the name of default must and should Function.
- B) main() is same as int main()
- C) By default, return 0 is added as the last statement of a function without specific return type.
- D) All the above

**Answer: D**

4) A function which calls itself is called a \_\_\_ function.

- A) Self Function
- B) Auto Function
- C) Recursive Function
- D) Static Function

**Answer: C**

5) What is the output of C Program with Functions.?

```
int main()  
{  
    void show()  
    {  
        printf("HIDE");  
    }  
  
    show();  
  
    return 0;  
}
```

- A) No output
- B) HIDE
- C) Compiler error
- D) None of the above

**Answer: B**

Explanation: Notice that show() function is defined inside main() function. It will not produce a compile error. But, it is not recommended to define a FUNCTION INSIDE A FUNCTION. DO NOT DO.

6) What is the output of C Program with functions.?

```
void show();  
  
int main()  
{  
    show();  
    printf("ARGENTINA ");  
    return 0;  
}  
  
void show()  
{  
    printf("AFRICA ");  
}
```



- A) ARGENTINA AFRICA
- C) ARGENTINA

- B) AFRICA ARGENTINA
- D) Compiler error

**Answer: B**

Explanation: First show() function is called. So it prints AFRICA first.

**7) What is the output of C Program with functions.?**

```
int main()
{
    show();
    printf("BANK ");
    return 0;
}
void show()
{
    printf("CURRENCY ");
}
```

- A) CURRENCY BANK
- C) BANK
- B) BANK CURRENCY
- D) Compiler error

**Answer: D**

Explanation: Yes. Compiler error. Before calling the show(); function, its Function Prototype should be declared before outside of main() and before main().

```
void show();
int main()
{
    show();
    printf("BANK ");
    return 0;
}
```

**8) How many values can a C Function return at a time.?**

- A) Only One Value
- C) Maximum of three values
- B) Maximum of two values
- D) Maximum of 8 values

**Answer: A**

Explanation: Using a return val; statement, you can return only one value.

**9) What is the output of a C program with functions.?**

```
void show();

void main()
{
    show();
    printf("RAINBOW ");

    return;
}
```

```
void show()
{
    printf("COLOURS ");
}
```

- A) RAI NBOW COLOURS
- C) COLOURS
- B) COLOURS RAINBOW
- D) Compiler error

**Answer: B**

Explanation: VOID functions should not return anything. RETURN; is returning nothing.

1. First void main() return; nothing. Still it is valid.

2. Second void show() function is NO RETURN statement. It is also valid.

**10) What is the output of C Program.?**



13) What are types of Functions in C Language.?

- A) Library Functions                      B) User Defined Functions  
C) Both Library and User Defined        D) None of the above

**Answer: C**

14) What is the output of C program with functions.?

```
int show();
```

```
void main()
{
    int a;
    a=show();
    printf("%d", a);
}
```

```
int show()
{
    return 15.5;
    return 35;
}
```

- A) 15.5            B) 15            C) 0            D) Compiler error

**Answer: B**

Explanation: It is perfectly Okay to return a float number 15.5 as an Integer inside int show() function. 15.5 is demoted to integer as 15 and returned.

15) What is the output of C Program.?

```
int myshow(int);
```

```
void main()
{
    myshow(5);
    myshow(10);
}
```

```
int myshow(int b)
{
    printf("Received %d, ", b);
}
```

- A) Received 5, Received 10,                      B) Received 10, Received 5,  
C) Received 0, Received 0,                      D) Compiler error

**Answer: A**

Explanation: Notice the function prototype declaration int myshow(int). If you declare wrong either Compiler warning or error is thrown. myshow(5) passes number 5. 5 is received as variable int b.

16) What is the output of C Program with functions and pointers.?

```
int myshow(int);
```

```
void main()
{
    int a=10;
    myshow(a);
    myshow(&a);
}
```

```
int myshow(int b)
{
    printf("Received %d, ", b);
}
```

- A) Received 10, Received 10,
- B) Received 10, Received RANDOMNumber,
- C) Received 10, Received RANDOMNumber, with a compiler warning
- D) Compiler error

**Answer: C**

Explanation: a is 10. &a is the address of the variable a which is a random memory location. To receive an address, int myshow(int b) should be rewritten as int myshow(int \*k).

**17) What is the output of C Program with functions and pointers.?**

```
int myshow(int *);
```

```
void main()
{
    int a=10;
    myshow(&a);
}
```

```
int myshow(int *k)
{
    printf("Received %d, ", *k);
}
```

- A) Received RANDOMNumber,
- B) Received 10,
- C) Received 10,
- D) Compiler error

**Answer: C**

Explanation: It is called Passing a variable by reference. You are passing &a instead of a. Address of a or &a is received as int \*k. Observe the function prototype declaration before main(), int myshow(int \*).

**18) What is the output of C Program with functions and pointers.?**

```
void myshow(int *);
```

```
void main()
{
    int a=10;
    printf("%d ", a);
    myshow(&a);
    printf("%d", a);
}
```

```
void myshow(int *k)
{
    *k=20;
}
```

- A) 10 10
- B) 20 20
- C) 10 20
- D) Compiler error

**Answer: C**

Explanation: You passed &a instead of a into myshow(int) function. \*k=20 changes the valued of passed variable passed by reference.

**19) What is the output of C Program with functions.?**

```
void myshow(int);
```

```
void main()
{
    int a=10;
    printf("%d ", a);
    myshow(a);
    printf("%d", a);
}
```

```
}
```

```
void myshow(int k)
```

```
{
```

```
    k=20;
```

```
}
```

A) 10 10

B) 20 20

C) 10 20

D) Compiler error

**Answer: A**

Explanation: You passed variable a directly by value. myshow(a). k=20 will not actually change the variable a as variable k and variable a are completely different. It is called Pass By Value.

**20) Choose correct statements about C Language Pass By Value.**

A) Pass By Value copies the variable value in one more memory location.

B) Pass By Value does not use Pointers.

C) Pass By Value protects your source or original variables from changes in outside functions or called functions.

D) All the above

**Answer: D**

**Programs:**

Q1. Input user name and password from the keyboard and write a function to encrypt this password.

Q2. Write a recursive function to print first 10 Fibonacci numbers.

Q3. Write a recursive function to check the given number is prime or not.

Q4. Write a function to print factorial of a given number.

Q5. Write a function to check the given string is palindrome or not?

\*\*\*\*\*

## SECTION - 6

\*\*\*\*\*

### Structures in C

C arrays allow you to define type of variables that can hold several data items of the same kind but **structure** is another user defined data type available in C programming, which allows you to combine data items of different kinds.

Structures are used to represent a record, Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book:

- Title
- Author
- Subject

- Book ID

## ❖ Defining a Structure

To define a structure, you must use the **struct** keyword. The struct statement defines a new data type, with more than one member for your program. The format of the struct statement is this:

```
struct [structure tag]
{
    member definition;
    member definition;
    ...
    member definition;
} [one or more structure variables];
```

The **structure tag** is optional and each member definition is a normal variable definition, such as `int i;` or `float f;` or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure:

```
struct Books
{
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
} book;
```

## 7. Accessing Structure Members

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use **struct** keyword to define variables of structure type. Following is the example to explain usage of structure:

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

struct Books
{
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};

void main( )
{
    /* Declare Book1 of type Book */
    struct Books Book1 = {"C Programming","Herbert","C Tutorial",6495407};

    /* Declare Book2 of type Book */
    struct Books Book2 = {"Telecom","Bertz","Tele Billing",6495700};

    /* print Book1 info */
    printf( "Book 1 title   : %s\n", Book1.title);
    printf( "Book 1 author  : %s\n", Book1.author);
    printf( "Book 1 subject : %s\n", Book1.subject);
    printf( "Book 1 book_id  : %d\n", Book1.book_id);

    /* print Book2 info */
    printf( "Book 2 title   : %s\n", Book2.title);
    printf( "Book 2 author  : %s\n", Book2.author);
```

```

printf( "Book 2 subject : %s\n", Book2.subject);
printf( "Book 2 book_id : %d\n", Book2.book_id);

}

```

When the above code is compiled and executed, it produces following result:

```

Book 1 title    : C Programming
Book 1 author   : Herbert
Book 1 subject  : C Tutorial
Book 1 book_id  : 6495407
Book 2 title    : Telecom
Book 2 author   : Bertz
Book 2 subject  : Tele Billing
Book 2 book_id  : 6495700

```

### Program:

**Q1.** Create a database of 10 student in C having the following attributes: *Roll\_no, Stud\_name, Age, City* And print details of those students who live in **Allahabad**.

\*\*\*\*\*

## SECTION - 7

\*\*\*\*\*

### File Handling

- A file is a collection of bytes stored on a secondary storage device, which is generally a disk of some kind.
- A file can be a text file or a binary file depending upon its contents.
- Through file handling, one can perform operations like create, modify, delete etc on system files.
- File I/O can be performed on a character by character basis, a line by line basis, a record by record basis or a chunk by chunk basis.
- Special functions have been designed for handling file operations. The header file **stdio.h** is required for using these functions.

#### Opening a File

Before we perform any operations on a file, we need to identify the file to the system and open it. We do this by using a file pointer. The type *FILE* defined in *stdio.h* allows us to define a file pointer. Then you use the function *fopen()* for opening a file. Once this is done one can read or write to the file using the *fread()* or *fwrite()* functions, respectively. The *fclose()* function is used to explicitly close any opened file.

#### fopen()

Before we can read (or write) information from (to) a file on a disk we must open the file. To open the file we have called the function **fopen()**.

**fopen()** returns the address of the file, which we have collected in the file pointer called **fp**. We have declared **fp** as

**FILE \*fp ;**

Syntax of fopen();->

**fp = fopen(" file name " , "Opening Mode");**

This function accepts two arguments as strings. The first argument denotes the name of the file to be opened and the second signifies the mode in which the file is to be opened. The second argument can be any of the following:

**The mode can be :**

- **'r'** : Open text file for reading. The stream is positioned at the beginning of the file.
- **'r+'** : Open for reading and writing. The stream is positioned at the beginning of the file.
- **'w'** : Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
- **'w+'** : Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.
- **'a'** : Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
- **'a+'** : Open for reading and appending (writing at end of file). The file is created if it does not exist. The initial file position for reading is at the beginning of the file, but output is always appended to the end of the file.

**fclose( )**

- The *fclose()* function is used for closing opened files. The only argument it accepts is the file pointer.
- If a program terminates, it automatically closes all opened files.

Syntax :-> fclose(FILE \*)

**Ex:- fclose( fp );**

**fputc( ) :->** Writes a single character to a file.

**Syntax:- fputc( char , FILE \*);**

**Ex:-> char ch='a';**

**fputc(ch,fp);**

**fgetc( ) :- >** Reads a single character from a file.

**Syntax:- char fgetc(FILE \*);**

**Ex:- > char ch=fgetc(fp);**

**fputs( ) :->** Writes a string in to file.

**Syntax:-> fputs(const char \*,FILE \*)**

**Ex:- fputs("bhanu pratap ",fp);**

**fgets ( ) :à** Reads a string from a file.

**Syntax:->char \* fgets(char \*, int, FILE \*)**

**Ex:-> char ch[50];**

**ch=fgets(fp);**

**fread( ) :->** Read data from a file

**fwrite( ):->** Writes data to a file

**ftell( ) :**

Function *ftell()* returns the current position of the file pointer in a stream. The return value is 0 or a positive integer indicating the byte offset from the beginning of an open file. A return value of -1 indicates an error.



**long int ftell(FILE \*fp);**

**fseek( )**

This function positions the next I/O operation on an open stream to a new position relative to the current position.

```
int fseek(FILE *fp, long int offset, int origin);
```

Here *fp* is the file pointer of the stream on which I/O operations are carried on, *offset* is the number of bytes to skip over. The *offset* can be either positive or negative, denoting forward or backward movement in the file. *origin* is the position in the stream to which the offset is applied, this can be one of the following constants :

SEEK\_SET : offset is relative to beginning of the file

SEEK\_CUR : offset is relative to the current position in the file

SEEK\_END : offset is relative to end of the file

**EOF:** It is a constant indicating that End Of File has been reached in a file.

**Program:**

**Q1:** WAP to Copy content of one file "test.txt" to another file "aa.txt"

**Q2.** WAP to count number of lines, number of spaces and number of characters of a file.

**Q3.** WAP to encrypt the content of a file and store it into another file.