# PROGRAMMING IN ANSI C REVIEW QUESTION SOLUTION

E BALAGURUSAMY (3rd edition)

MOHAMMAD ABIR REZA

**Department of Computer Science & Engineering, Comilla University**

# Speech of the Author

I am Mohammad Abir Reza. I am studying Computer Science and Engineering at Comilla University. I made a solution of "PROGRAMMING IN ANSI C" written by

E Balagurusamy, but there were some mistakes in my previous edition . So I tried to recover those mistakes in this 3rd edition. If there is any mistake you can help me by informing and I will try to recover the mistake in next edition .You can mail me to the following address abir.cse.cou@gmail.com

Now this is time for thanking my friend Anon.

Hi Anon,

Special thanks to you for helping me from the beginning. I could not do this job without your help and hope you will help me in my next projects also .
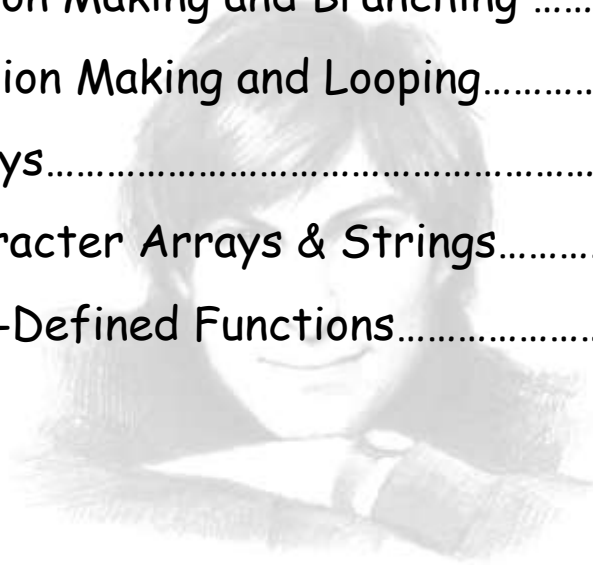
Mohammad Abir Reza

Department of Computer Science & Engineering

Comilla University

# Contents

STEVE JOBS

DESIGNER OF NEW WORLD

# Overview of C

## 1.11   Why and when do we use the #define directive?

A #define is a *preprocessor* compiler directive. We often use certain unique constants in a program. These constants may appear repeatedly in a number of places in the program. We face two problems in the subsequent use of such programs. These are 1. problem in modification of the program and 2. problem in understanding the program. Assignment of such constants to a *symbolic name* frees us from these problems. So if we want to define values to a symbolic constant and to define any statements which uses more frequently in the program we use the #define directive. Whenever a symbolic name is encountered, the compiler substitutes the value associated with the name automatically. To change the value, we have to simply change the definition.

A constant is defined as follows:

#define  symbolic-name  value of constant

## 1.12   Why and when do we use the #include directive?

C programs are divided into modules or functions. Some functions are written by users, like us, and many others are stored in the C library. Library functions are grouped category-wise and stored in different files known as *header* files. If we want to access the functions stored in the library, it is necessary to tell the compiler about the files to be accessed.

This is achieved by using the preprocessor directive #include as follows:

#include<filename>

Filename is the name of the library file that contains the required function definition. Preprocessor directives are placed at the beginning of a program.

## 1.13. What does void main (void) mean?

The main is a part of a every C program. C permits different forms of main statement. Following forms are allowed.

main()

int main()

void main()

main(void)

void main void()

int main(void)

The empty pair of parentheses indicates that the function has no arguments. This may be explicitly indicated by using the keyword **void** inside the parentheses. We may also specify the keyword **int** or **void** before the word main. The keyword **void** means that the function does not return any information to the operating system and **int** means that the function returns an integer value to the operating system. When **int** is specified, the last statement in the program must be "return 0".

## 1.15  Why do we need to use comments in programs?

It is a good practice to use comment lines in the beginning to give information such as name of the program, author, date, etc. Comment characters are also used in other lines to indicate line numbers. The generous use of comments inside a program cannot be overemphasized. Since comments do not affect the execution speed and the size of a compiled program, we should use them liberally in our programs. They help the programmers  and other users in understanding the various functions

and operations of a program and serve as an aid to debugging and testing. Judiciously inserted comments not only increase the readability but also help to understand the program logic.

## 1.16 Why is the look of a program is important?

Look of a program is very important. If anyone is working in an IT industry, as his or her program is going to be referred by many people. When they read the program they should get clear idea about what that program is written for and they should not even need to read all the program. For that when anyone writes a program one must use some coding conventions, such as: standard variable name format, function name format etc. A proper indentation of braces and statements is very important for the good look of a program. This is very important for debugging and testing the program.

## 1.17 Where are blank spaces permitted in a C program?

C is a free form language. Blank spaces are completely valid at the beginning, middle or end of a line except in keywords and identifiers. Spaces are permitted in stings which are enclosed in double quotes (" ").

## 1.18 Describe the structure of a C program.

| Documentation Section |
|---|
| Link Section |
| Definition Section |
| Global Declaration Section |

| main () Function Section |
| :--- |
| { |
| <table><tr><td>**Declaration part**</td></tr><tr><td>**Executable part**</td></tr></table> |
| } |

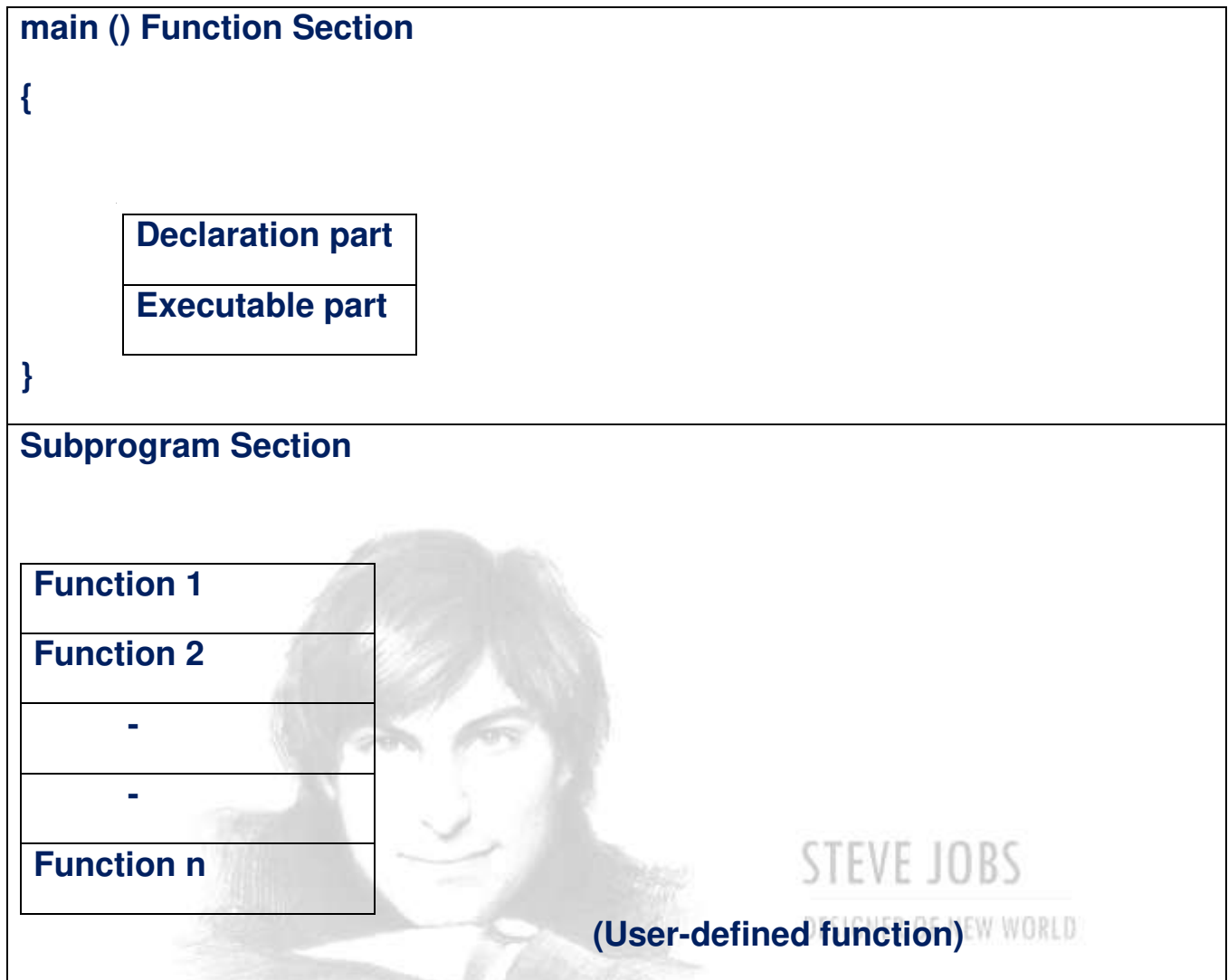| **Subprogram Section** |
| :--- |
| <table><tr><td>**Function 1**</td></tr><tr><td>**Function 2**</td></tr><tr><td>-</td></tr><tr><td>-</td></tr><tr><td>**Function n**</td></tr></table> **(User-defined function)** |

Fig. *An overview of a C program*

A    C program may contain one or more sections:

The documentation section consists of a set of common lines giving the name of the program, the author and other details, which the programmer would like to use later.

The link section provides instructions to the compiler to link functions from the system library.

The definition section defines all symbolic constants.

There are some variables that are used in more than one function. Such variables are called *global* variables and are declared in the *global* declaration

section that is outside of all the functions. This section also declares all the user-defined functions.

Every C program must have one **main()** function section. This section contains two parts, declaration part and executable part. The declaration part declares all the variables used in the executable part. There is at least one statement in the executable part. These two parts must appear between the opening and the closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function section is the logical end of the program. All statements in the declaration and executable parts end with a semicolon (;).

The subprogram section contains all the user-defined functions that are called in the **main** function. User-defined functions are generally placed immediately after the **main** function, although they may appear in any order. All sections, except the main function section may be absent when they are not required.

## 1.19 Describe the process of creating and executing a C program under UNIX system.

### Creating the program:

Once we load the UNIX operating system into the memory, the computer is ready to receive program. The program must be entered into a file. The file name can consist of letters, digits and special characters, followed by a dot and a letter *c*. Examples of valid file names are:

*hello.c , program.c*

The file is created with the help of a *text editor*, either **ed** or **vi**. The command for calling the editor and creating the file is

**ed** *filename*

If the file existed before, it is loaded. If it does not yet exist, the file has to be created so that it is ready to receive the new program. Any corrections in the program are done under the editor.

When the editing is over, the file is saved in the disk. It can then be referenced any time later by its file name. The program that is entered into the file is known the source program, since it represents the original form of the program.

**Executing the program:**

Execution is a simple task. The command

<div align="center">a.out</div>

would load the executable object code into the computer memory and execute the instructions. During execution, the program may request for some data to be entered through the keyboard. Sometimes the program does not produce the desired results. Perhaps, something is wrong with the program logic or data. Then it would be necessary to correct the source program or the data. In case the source program is modified, the entire process of compiling, linking and executing the program should be repeated.

## 1.20 How do we implement multiple source program files?

To compile and link source program files, we must append all the files names to the **cc** command.

<div align="center"><strong>cc</strong> <em>filename</em>-1.c .<strong>...</strong> <em>filename</em>-n.c</div>

These files will be separately compiled into object files called

<div align="center"><strong>filename-i.o</strong></div>

and then linked to produce an executable program file **a.out** as shown in figure.

It is also possible to compile each file separately and link them later. For example, the commands

**cc   -c** mod1.c

**cc   -c** mod2.c

will compile the source files *mod1.c* and *mod2.c* into object files *mod1.o* and *mod2.o*. They can be linked together by the command

**cc** mod1.o  mod2.o

we may also combine the source files and object files as follows:

**cc** mod1.c  mod2.o

Only mod1.c is compiled and then linked with the object file mod2.o. This approach is useful when one of the multiple source files need to be changed and recompiled or an already existing object files is to be used along with the program to be compiled.
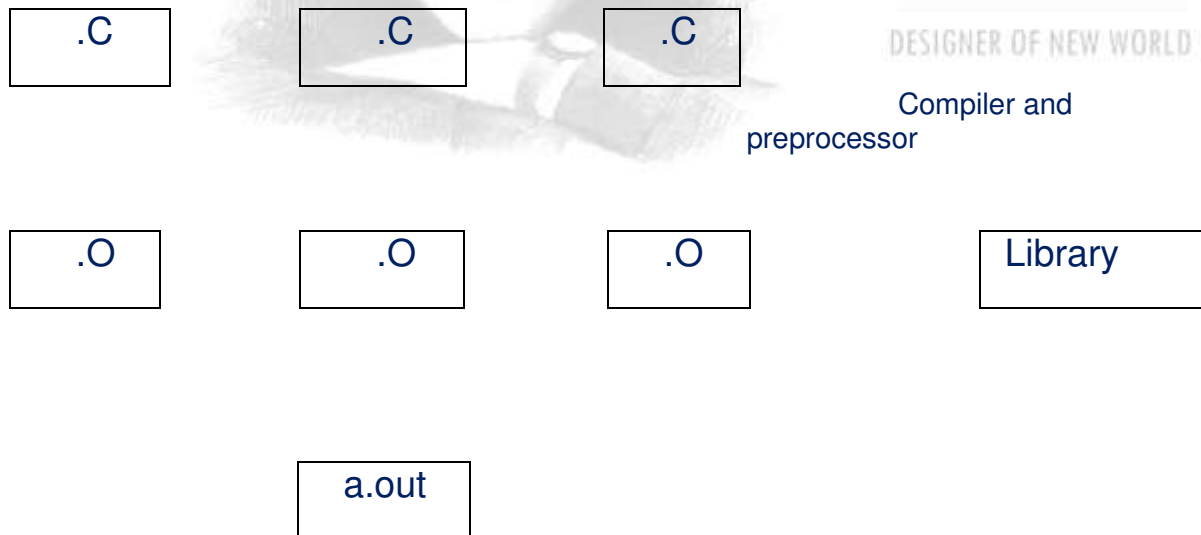
| .C | .C | .C |
|----|----|----|

Compiler and preprocessor

| .O | .O | .O | Library |
|----|----|----|---------|

| a.out |
|-------|

Fig. *Compilation of multiple files*

# Chapter -02

# Constants, Variables and Data Types

2.1 State whether the following statements are  true or false:

(a)   Any valid printable ANSII character can be used in an identifier. ( False )

(b)   All variables must be given a type when they are declared. ( True )

(c)    Declarations can appear anywhere in a program. ( False )

(d)    ANSI C treats the variable name and Name to be same. ( False )

(e)   The underscore can be used anywhere in an identifier. ( True )

(f)    The keyword void is a data type in C. ( True )

(g)   Floating point data constants, by default, denote float type values.( False )

(h)   Like variables, constants have a type. ( True )

(i)    Character constants are coded using double quotes. (False )

(j)    Initialization is the process of assigning a value to a variable at the time of declaration.   ( true )

(k)    All static variables are automatically initialized to zero. ( True )

(l)    The scanf function can be used to read only one value at a time. ( False )

2.2  Fill in the blanks with appropriate words:

(a)The keyword ……………..can be used to create a data type identifier.

**Answer: int**

(b) …………… is the largest value that an unsigned short int type variable can store.

**Answer: 255**

(c) A global variable is also known as ……………variable.

**Answer: external**

(d) A variable can be made constant by declaring it with the qualifier ……………. at the time of initialization.

**Answer: constant**

## 2.3 What are trigraph characters? How are they useful?

Trigraph characters is one kinds of character which consists of three characters ( Two question marks and followed by another ).

Some keyboard does not support some characters. But we can use them by trigraph characters.

**For example**: If a keyboard does not support square brackets, we can still use them in a program using the trigraph ??( and ??) .

## 2.4 Describe the four basic data types. How could we extend the range of values they represent?

The basic four data types are:

(1) Char

(2) Int

(3) Float

(4) Void


We cannot extend the range of character.

We could extend the range of integer by using long before integer.

We can extend the range of float by using double. To extend the precision further we may use long double.

**Integer Types:** Integers are whole numbers with a range of values supported by a particular machine. Generally, integers occupy one word of storage, and since the word sizes of machines vary (tipically, 16 or 32 bits) the size of an integer that can be stored depends on the computer. If we use a 16 bit word length, the size of the integer value is limited to the range -32768 to +32767 (that is , $-2^{15}$ to $+2^{15}$-1). A signed integer uses one bit for sign and 15 bits for the magnitude of the number. Similarly, a 32 bit word length can store an integer ranging from -2,147,483,648 to 2,147,483,647.

In order to provide some control over the range of numbers and storage space, C has three classes of integer storage, namely **short int**, **int**, and **long int**, in both **signed** and **unsigned** forms. ANSI C defines these types so that they can be organized from the smallest to the largest. For example, **short int** represents fairly small integer values and requires half the amount of storage as a regular **int** number uses. Unlike signed integers, unsigned integers use all the bits for the magnitude of the number and are always positive. Therefore, for a 16 bit machine, the range of unsigned integer numbers will be from 0 to 65,535.We declare **long** and **unsigned** integers to increase the range of values. The use of qualifier **signed** on integers is optional because the default declaration assumes a signed number.

## Floating point Types:

Floating point (or real) numbers are stored in 32 bits (on all 16 bit and 32 bit machines), with 6 digits of precision. Floating point numbers are defined in C by the keyword **float**.

## Double Types:

When the accuracy provided by a float number is not sufficient, the type double can be used to define the number. A double data type number is not sufficient, the type double can be used to define the number. A double data type number uses 64 bits giving a precision of 14 digits. These are known as double precision numbers. Double type represents the same data type that float represents, but with a greater precision. To extend the precision further, we may use long double which uses 80 bits.

## Character Types:

A single character can be defined as **character (char)** type data. Characters are usually stored in 8 bits (one byte) of internal storage. The qualifier **signed** or **unsigned** may be explicitly applied to char. While **unsigned chars** have values between 0 and 255, **signed chars** have values from -128 to 127.

## How can we extend the range of values they represents:

We can extend the range of values of integer types by using the qualifier **long** and **unsigned** before **int**.

We can extend the range of values of floating type data by using double data type. To extend the precision further we may use the qualifier **long** before double.

The qualifier **unsigned** may be explicitly applied to char to extend the range of  character  data type.

## 2.5 What is an unsigned integer constant? What is the significant of declaring a constant unsigned?

An unsigned constant will not have any sign means, it will be having positive values only.

for example if we use
    short int a;
the range of a will be from -32,768 to +32,767(means you can enter any no between this range) .
On the other hand , if we use

    unsigned short int a;
the range of a will be doubled and it will have only positive values the range will be 0 to +65,535.
WHY WE USE IT?

sometimes we don't need variables to take negative values
for example-

    int age;
Here age will not be in negative. Moreover we can double the range and store large number by using unsigned .

## 2.6 Describe the characteristics and purpose of escape sequence characters.

C supports some special back slash character constants, that are used in output function. This characters are known as escape sequence characters. For example, the symbol "\n" stands for new line character.

**Characteristics :**

  a. Although it consists of two characters, it represents single character.

  b. Every combination starts with backslash (\).

  c. They are nonprinting characters.
d. It can also be expressed in terms of octal digits or hexadecimal sequence.
e. Each escape sequence has unique ASCII value.
f. Escape sequence in character constants and string literals are replaced by their equivalent and then adjacent string literals are concatenated .

  **Purpose:**

  1. In a program we use it for new line.

  2.In a program we use it for horizontal tab.

  3. We can represent any member of the execution character set by an *escape sequence.*

  4. Escape sequences are primarily used to put nonprintable characters in character and string literals.

## 2.7What is a variable and what is meant by "value" of a variable?

A variable is a data name that may be used to store a data value. Unlike constants that remains unchanged during the execution of a program. A variable can take different values at different times during execution, just like character, int, float and double.

## 2.8 How do variables and symbolic names differ?

A variable may be used to store data value. A variable may take different values at different times during execution of a program. Variables need to declare at the beginning of the body but after the main.

Symbolic names are unique constants. These constants may appear in a number of place in the program. Symbolic names need to be defined at the beginning of a program.

## 2.9 State the difference between the declaration of a variable and the definition of a symbolic name?

Variables are declared at the beginning of the body but after the main. All variables must be declared before they can appear in executable statements. The syntax for declaring a variable is as follow:

**Data-type v1,v2,……..vn;**

v1, v2,……..vn are the names of variables. For example, valid declarations are

**int count;**

**int number,total;**

**float ratio;**

*Symbolic names* need to be defined at the beginning of a program.   A symbolic name constant is defined as follows:

#define  SYMBOLIC-NAME  value of constant

**Valid example of constant definations are:**

**#define STRENGTH    100**

**#define PASS MARK  5**

Declaration statement of a variable must end with a semicolon. Definition of a *symbolic name* must not end with a semicolon. A variable may take different values at different times during execution. *Symbolic name* should not be assigned any other value within the program by using an assignment statement.   *Symbolic names* are not declared for data type. Its data type depends on the type of constant.   But we must declare the data type of a variable. *Symbolic names* are written in CAPITALS to visually distinguish them from the normal variable names, which are written in lowercase letters. This is only a convention not a rule. An instance of an object is created when a variable is declared. Definition of a *symbolic name* just defines a name that can be used in the program.

## 2.10 What is initialization? Why it is important?

The process of giving initial values to variables is called initialization.
C allows ordinary variables, structures, unions and arrays to be given initial values in their definitions.  There are basically two sorts of initialization: at compile time, and at run time. Example of initialization :

**int a=100;**

**char name   = 'x';**

**float number =75.84 ;**

This statement initializes the variable a to 100. External and static variables are initialized to zero by default. Automatic variables that are not initialized explicitly will contain garbage.

C permits the initialization of more then one variable using multiple assignment operators. For example the following statements are valid.

p=q=s=0;

x=y=z=MAX;

**Importance of initialization:**

Initialization is so important because of the following:

Initialization helps us to know the value of a variable. If we do not initialize a variable it will contain garbage value. The variables that are used in expressions(on the right side of equal(=) sign of a computational statement) must be assigned values before they are encountered in the program. Initialization helps us to understand a program. An uninitialized variable holds the value of the previous memory contents(sometimes). This becomes a problem when using pointers which should be initialized to null until we are ready to use them because they point at anything.
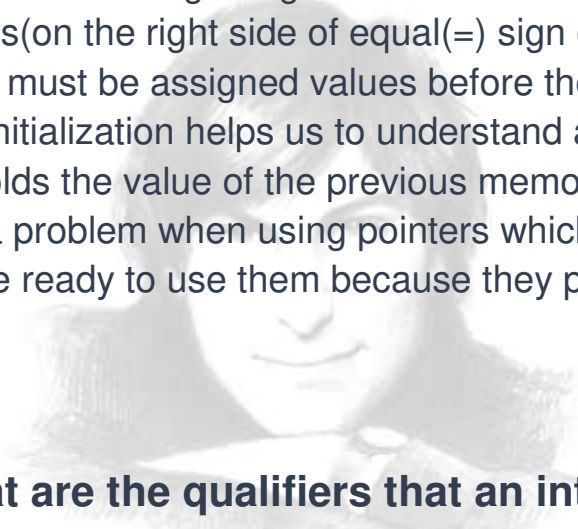
STEVE JOBS

DESIGNER OF NEW WORLD

## 2.11 What are the qualifiers that an int can have at a time?

An **int** can have **short**, **long**, **signed**, **unsigned**, **unsigned short**, **unsigned long** qualifiers at a time.

A signed  int can take a value between -32768 to 32767 and an unsigned int can take a value 0 to 65535.

## 2.12  A programmer would like to use the word DPR to declare all the double-precision floating point values in his program. How could he achieve this?

He can put a line like this before the main function in the program:
**#define DPR double**
From that point on, he can use DPR in place of double wherever he wants.
For example : he can write
**DPR  my_double_variable;**
instead of
**double my_double_variable;**

Or he can declare DPR  as typedef data type. Such as
**typedef  double DPR;**
now DPR  can be later used to declare double type variables as follows:
DPR a; (where a is a variable of double data type).

## 2.13  What are enumeration variables? How are they declared? What is the advantage of using them in a program?

Enumerated is a user-defined data type provided by ANSI standard. It is defined as follows:

**enum identifier {member 1, member 2, … member n};**

The  "identifier" is a user-defined enumerated data type which can be used to declare variables that can have one values enclosed within the braces (known as enumeration constants). These variables are called enumeration variables.

### Declaration:

An enumeration is user-defined data type. Its members are constants that are written as identifiers, though they have signed integer values. These constants represent valued that can be assigned to corresponding enumeration variables.

In general terms, an enumeration may be defined as

enum identifier {member 1, member 2, … member n};

where  enum is a required keyword; identifier is a name that identifies enumerations having this composition; and member 1, member 2, … member n represent the individual identifiers that may be assigned to variables of this type. The member names must differ from one another, and they must be distinct from other identifiers whose scope is the same as that of enumeration.

Once the enumeration has been defined, corresponding enumeration variables can declared as

enum identifier v1, v2, … vn;

where  enum is a required keyword, identifier is the name that appeared in the enumeration definition, and v1, v2, … vn are enumeration variables of type identifier.

The enumeration definition can be combined with variable declarations, as indicated below.

enum identifier {member 1, member 2, … member n} v1, v2, … vn;

The identifier is optional in this situation.

**Advantage:**

1. Enumeration helps to provide self-documenting code.

2. Enumeration helps to clarify the structure of a program.

3. Enumerated constants are generated automatically by the compiler.

4. Enumeration helps programs to become more readable and thus can be understood better by others who might have to update these programs later.

5. Enumeration helps to improve the debugging capacity of a program.

## 2.14 Describe the purpose of the qualifiers constant and volatile.

**Purpose of qualifier const:**

We may like the value of certain variables to remain constant during the execution of a program. We can achieve this by declaring the variable with the qualifier **const** at the time of initialization. Example

**const** int class_size=40;

const is a data type qualifier defined by ANSI standard. This tells the compiler that the value of the int variable class_size must not be modified by the program. However, it can be used on the right hand side of an assignment statement like any other variable.

**Purpose of qualifier volatile:**

ANSI standard defines qualifier **volatile** that can be used to tell explicitly the compiler that a variable's value may be changed at any time by some external sources (from outside the program). For example:

**volatile int** date;

The value of **date** may be altered by some external factors even if it does not appear on the left-hand side of an assignment statement. When we declare a variable as **volatile**, the compiler will examine the value of the variable each time it is encountered to see whether any external alteration has changed the value.

The value of a variable declared as **volatile** can be modified by its own program as well. If we wish that the value must not be modified by the program while it may be altered by some other process, then we may declare the variable as both **const** and **volatile** as shown below:

**volatile const int** location=100;

## 2.15 When dealing with very small or very large numbers, what steps would you like you take to improve the accurancy of the calculation?

When dealing with very small numbers I will like to use the qualifier **short** before data type **int** if the number is integer. Or if the number is floating point type I will use **float** data type to improve the accuracy of the calculations.

When dealing with very large number if the number is a integer I will use the qualifier **long** before **int**. If the number is a positive integer I may use the qualifier **unsigned** or **unsigned long**. If the number is floating point type I will use data type **double** or use the qualifier **long** before **double** to improve the accuracy of the calculations.

## 2.16  Which of the following are invalid constants and why?

0.0001

Answer:  valid

5x1.5

Answer:  Invalid

Reason: Exponent must be an integer.

99999

Answer: Valid

Reason: Long integer.

+100

Answer:  valid

75.45E-2

Answer:  Valid

-45.6

Answer:  Valid

"15.75"

Answer:  Invalid

Reason: "" sign is not permitted.

-1.79e+4

Answer: valid

0.00001234

Answer:  Valid

## 2.17 Which of the following are invalid variable Question and why?

Minimum

Answer:  valid

First.name

Answer:  Invalid

Reason:   . Sign is not permitted.

N1+n2

Answer:  Invalid

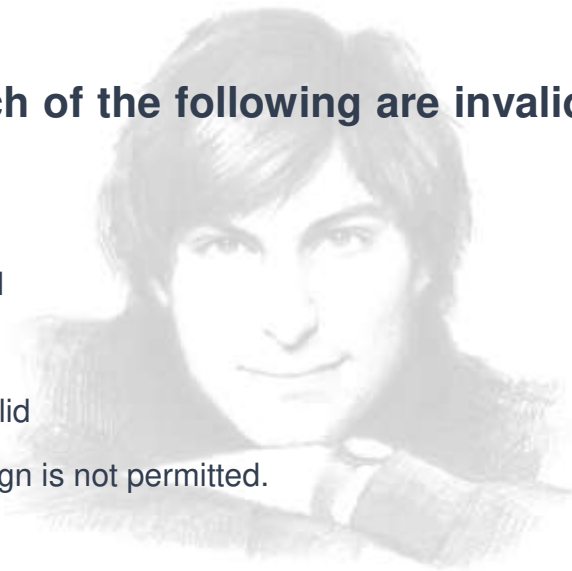Reason: + sign is not permitted.

&name

Answer:  Invalid

Reason:  & is not permitted.

Doubles

Answer:   Valid

Reason: Keyword may be a part of variable name.

3rd_row

Answer:  Invalid

Reason: First character must be a letter or underscore.

n$

Answer:   Invalid

Reason: Dollar sign is not permitted.

Row1

Answer:  Valid

Float

Answer:   Invalid

Reason: float is a keyword.

Sum Total

Answer:  Invalid

Reason: White space is not permitted.
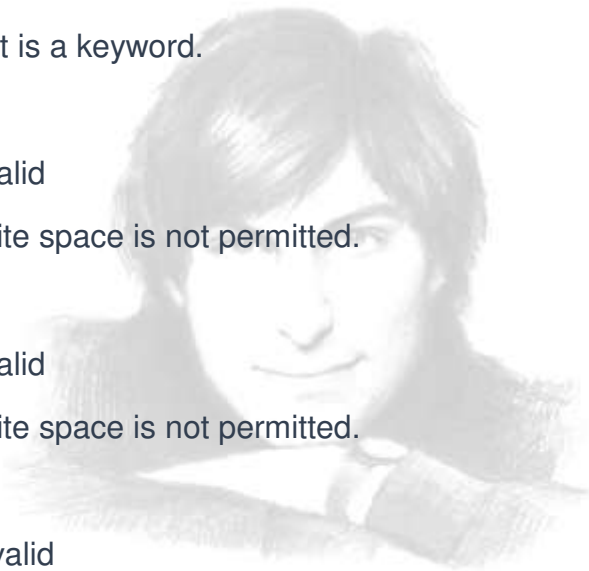
Row Total

Answer:  Invalid

Reason: White space is not permitted.

Column total

Answer:   Invalid

Reason: White space is not permitted.

STEVE JOBS

DESIGNER OF NEW WORLD

2.18 Find errors, if any, in the following declaration statements.

Int x;

Answer : Error. It should be : **int x;**

float letter,DIGIT;

Answer: No Error.

double=p,q;

Answer: Error. It should be : **double p,q;**

exponent alpha,beta;

Answer : Error.

m,n.z:INTEGER

Answer : Error

short char c;

Answer : Error.

long int m;count;

Answer : Error. It should be : **long int m, count;**

long float temp;

Answer : Error.  long float is not valid.

2.19 What should be the value of x after execution of the following statement?

  int x,y=10;

  char z='a';

  x=y+z;

Answer: 107

**Mohammad Abir Reza [CSE- 5TH BATCH]**
**Department of Computer  Science & Engineering**
**Comilla University**

# Chapter : 03

# Operators and Expressions

3.3 Given the statement

Int a=10, b=20 ,c ;

Determine whether each of the following statements are true or false.

a . The statement a =+10, is valid.

Answer : False.

b. The expression a+ 4/6 *6/2 evaluates to 11 .

Answer : False.

c. The expression b+ 3/2* 2/3 evaluates to 20 .

Answer : True .

d. The statement a+=b; gives the values 30 to a and 20 to b.

Answer : True .

e. The statement ++a++ ; gives the value 12 to a

Answer : False.

f. The statement a=1/b ; assigns the value 0.5 to a

Answer : False.

3.4 Declared **a** as *int* and b as *float*, state whether the following statements are true or false.

a. The statement  a=1/3 + 1/3 + 1/3 ;  assigns the value  1 to a.

Answer : False.

b.The statement  b=1.0/3.0 + 1.0/3.0 + 1.0/3.0 ; assigns a value 1.0 to b.

Answer : True.

c. The statement   b=1.0/3.0 *3.0 gives a value 1.0 to b.

Answer : True.

d. The statement  b=1.0/3.0 + 2.0/3.0 assigns a value  1.0 to b.

Answer : True.

e.  The statement  a=15/10.0 + 3/2 ; assigns a value 3 to a.

Answer : False.

3.5 Which of the following arithmetic expressions are true ?

a. !(5+5 >=10)

Answer : False.

b. 5+5==10 || 1+3 ==5

Answer : true .

c. 5>10 || 10<20 && 3<5

Answer : true .

d. 10 != 15 && !(10<20)  || 15>30

Answer : False .

## 3.6 Which of the following arithmetic expressions are valid ? If valid , give the value of the expression ; otherwise give reason .

(a)25/3  %2

Answer:  0

Because the expression '25/3' evaluates to 8. Because the number 3 is int type. and the expression '8%2' evaluates to 0.

(b) +9/4 + 5

Answer :  7

(c)7.5%3

Answer :  Not valid.

Invalid operands to binary % (have 'float' and 'int' type)

(d) 14%3 + 7%2

Answer :  3

(e) -14%3

Answer :  -2

(f)15.25 + -5.0

Answer : 10.250000

(g)(5/3)* 3 + 5%3

Answer :   5

(h) 21%(int)4.5

Answer  :  1

3.8 Identify unnecessary parentheses in the following arithmetic expressions.(red color is unnecessary)

(a) ((x-(y/5)+z)%8)+25

(b) ((x-y)*p)+q

(c) (m*n) +(-x/y)

(d) x/(3*y)

3.9 Find errors , if any , in the following assignment statements and rectify them .

a. x=y=z = 0.5,2.0,-5.75 ;

Answer : Error.

Correct answer : x=0.5, y=2.0 , z=-5.75 ;

b. m=++a *5 ;

Answer : No Error.

c.  y=sqrt (100) ;

Answer : No error.

d.  p*    =x/y ;

Answer : Error.(no space between * and =)

e.  s= /5 ;

Answer : Error.('/'  should be used before '=')

f.  a=  b++ - c*2

Answer : Error.(semicolon)

STEVE JOBS

DESIGNER OF NEW WORLD

3.10 Determine the value of each of the following logical Expressions  if  a=5 , b=10 and c= -6

a.      a>b && a<c

Answer : false .

b.      a<b && a>c

Answer : True.

c.     a==c || b>a

Answer : True.

d.     b>15 && c<0 || a>0

Answer : True.

e.      (a/2.0 ==0.0 && b/2.0  != 0.0) || c<0.0

Answer : True.
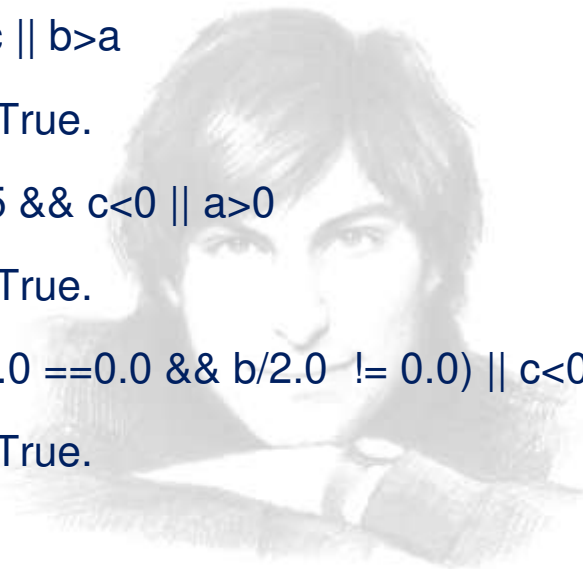
## 3.11 What is the output of the following program ?

```
main()
{
        char x;
        int y;
        x=100;
        y=125;
        printf("%c\n",x);
        printf("%c\n",y);
        printf("%d\n",x);
}
```

Output :

d

}

100

## 3.12 Find the output of the following program ?

```
main()
{
        int x =100;
        printf("%d/n ",10+ x++);
        printf("%d/n",10+  ++x);
}
```

Output :  110/n112/n

3.13 What is the output of the following program ?

```
main()
{
        int x=5, y=10 , z=10;

        x= y==z;

        printf("%d",x);

}
```

**Output:   1**

3.14 What is the output of the following program ?

```
main()
{
        int x=100,y=200;

        printf("%d ",(x>y)? x:y);

}
```

**Output : 200**

## 3.15 What is the output of the following program ?

```
main()

{

        unsigned x=1;

        signed  y=-1;


        if(x>y)

                printf("x>y");

        else

                printf("x<=y");

}
```

Did you expect this output ? Explain

**Output : x<=y**

Yes I expected this output.

In this program to evaluate if expression the signed char y= -1 is converted to **unsigned int** type. So the 'if ' condition becomes false and 'else ' part in executed. So the output is x<=y.

## 3.16 What is the output of the following program ? Explain the output .

```
main()

{

        int x=10;

        if(x=20)

                printf("true");
```

else

printf("fulse");

}

Output : true

Computer evaluates the expression first and then depending on whether the value of the expression (relation or condition) is 'true' (or non zero) or 'false'(zero), it transfers the control to a particular statement. In this case the expression x=20 evaluates to 20(non zero). So the expression is true and the statement after it is executed. So the output is TRUE .

## 3.17 What is the error in each of the following statement ?

(a) if(m==1 & n!=0)

printf("ok");

Answer : Error

CA:   if(m==1 **&&** n!=0)

printf("ok");

STEVE JOBS

DESIGNER OF NEW WORLD

(b) if(x=<5)

printf("jump");

Answer: Error

CA:  if(x**<=**5)

printf("jump");

## 3.18 What is the error , if any , in the following segment ?

int x=10;

float y=4.25;

x= y% x;

**Answer :**There is an error in the segment. The modulo division operator '%' cannot be used on floating point data. Because real operator can't use '%' .

## 3.19 What is printed when the following is executed ?

```
for(m=0;m<3;++m)
        printf("%d\n",(m%2) ?  m:m+2);
```

Output : 2

1

4

## 3.20 What is the output of the following segment when executed ?

```
int m= -14,n=3;
printf("%d\n",m/n* 10);
n=-n;
printf("%d \n",m/n*10);
```

**Output : -40**

**40**

**Mohammad Abir Reza [CSE- 5TH BATCH]**
**Department of Computer  Science & Engineering**
**Comilla University**

# Managing Input and Output Operations

## 4.3 Distinguish between the following pairs:

### (a) getchar and scanf functions:

1. getchar is a macro that takes character as an input from stdin . where as scanf is a function that takes any data type as an input from stdin.

2. scanf uses different format specifiers like(%d, %f, %c, %s etc) to take different data types as input. But getchar does not use any format specifiers it only takes character as input.

3. At a time scanf can receive multiple data types as inputs. But getchar can't do this.

4. As getchar is a macro thats why it helps to make program run faster .

5. getchar is an unformatted function means it does not make use of format specifiers where as scanf is a formatted function.

6. The getchar function accepts any character keyed in. This includes RETURN and TAB. scanf function cannot be used to read strings with blank spaces.

### (b) %s and %c specifications for reading.

%s specifier is used to read a string. But it stops reading that string when the first whitespace character is encountered. A whitespace character is either a space, a tab, or a newline.

%c specifier is used to read a single character.

### (c) %s and %[] specifications for reading.

%s specifier is used to read a string. But it stops reading that string when the first whitespace character is encountered. A whitespace character is either a space, a tab, or a newline.

%[] specifier is used to read a string which may include whitespace characters.

### (d) %g and %f specification for printing.

%g specifier is used to print data item as a floating-point value using either e-type or f-type conversion, depending on value. Trailing zeros and trailing decimal point will not be displayed.

%f specifier is used to print data item as a floating-point value without an exponent.

### (e) %f and %e specification for printing.

%f specifier is used to print data item as a floating-point value without an exponent.

%e specifier is used to print data item as a floating-point value with an exponent.

## 4.6 State errors, if any , in the following input statements

a.  scanf("%c %f %d",city,&price,&year);

Answer : Error .

b. scanf("%s %d", city,amount);

Answer :  Error .

c. scanf("%f %d",&amount,&year);

Answer :  No Error .

d. scanf(\n"%f", root);

Answer : Error .

Cause: **\n will remain into double quote.**

e.  scanf("%c %d %ld",*code, &count, root);

Answer : Error

cause : Here '*' is not allowed before 'code'  and  '&' will stay before 'root'

## 4.8  The variables count , price and city have the following values :

Count < ------  1275

Price < -------235.74

City < --------- Cambridge

Show the exact output that the following output statements will produce :

a. printf("%d  %f" , count , price);
   Answer : 1275        235.740005

b. printf("%2d\n  %f" , count , price);

Answer :

1275

235.740005

c. printf ("%d  %f",price,count);

Answer:  536870912        0.000000

d. printf ("%10d xxxx %5.2f" , count , price);

    Answer:    1275 xxxx    235.74

e. printf("%s",city);
   Answer :  Cambridge
f. printf("%-10d  % -15s" , count, city);

    Answer : 1275       cambridge


## 4.9  State what (if anything) is wrong with each of the following output statements :

a. printf("d.7.2%f",year,amount);

ANSWER: WRONG   7.2 SHOULD REMAIN AFTER THE %

b. printf("%-s,%c"\n,city,code);

Answer : wrong   comma(,) is not allowed after 's'    and  '\n' will stay between quotation.

c. printf("%f %d %s",price,count,city);

 Answer :  No wrong .

d. printf ("%c %d %f\n" , amount ,code,year);

 Answer : No wrong .

## 4.10  In response to the input statement

Scanf("%4d %*%d", &year, &code ,&count);

The following data is keyed in :

   19883745

What values does the computer assign to the variables year , code, and count ?

Answer :   year will give output 1988  and both code & count will produce garbage value.

## 4.11  How can we use  getcher()  function to multi character strings?

The getchar() function may be called successively to read the characters contained in a line of text thus we can use the getchar() function to read multi character strings. For example, the following program segment reads characters from keyboard one after another until the 'Return' key is pressed.

```
main()
char character;
character = ' ' ;
while (character != '\n')
{
        character = getchar();
}
-----------------------
-----------------------
```

## 4.12  How can we use putchar() function to multi character strings?

 The putchar() function may be called successively to output the characters contained in a line of text thus we can use the putchar() function to output multi character strings. For example, the following program segment output characters from keyboard one after another.

```
 main()
{
char str[] = "Hello World";
int i=0;
while(str[i] != '\0')
putchar(str[iii++]);
}
```

## 4.13  What is the purpose of scanf() function?

scanf() function is standard input function in C Programming language. This function is defined inside header file called 'stdio.h'. scanf() means scan formatted. Input data can be entered into computer from a standard input device by means of the C library function scanf(). This function can be used to enter any combination of numerical values, single characters and strings. The function returns the number of data items that have been entered successfully.

The general form of scanf() is as follows:

scanf("control string", arg1, arg2, …, argn);

The control string specifies the field format in which the data is to be entered and the arguments arg1, arg2, …, argn specify the address of location where the data is stored. Control string and arguments are separated by commas.

Control string (also known as format string) contains field specifications, which direct the interpretation of input data. It may include:

1. Field (or format) specifications, consisting of the conversion character %, a data type character (or type specifier), and an optional number, specifying the field width.

2. Blanks, tabs, or newlines.

Blanks, tabs and newlines are ignored. The data type character indicates the type of data that is to be assigned to the variable associated with the corresponding argument. The field width specifier is optional.

## 4.14 Describe the purpose of commonly used conversion characters in a scanf() function ?

The main purpose of commonly used conversion characters in a scanf() function is these indicate what type of data we take as intput.

Commonly used scanf() format codes are

| Code | Meaning |
|------|---------|
| %c | read a single character |
| %d | read a decimal integer |
| %e | read a floating point value |
| %f | read a floating point value |
| %g | read a floating point value |
| %h | read a short integer |
| %i | read a decimal, hexadecimal or octal integer |
| %o | read an octal integer |
| %s | read a string |
| %u | read an unsigned decimal integer |

| %x | read a hexadecimal integer |
|---|---|
| %[..] | read a string of word(s) |

The following letters may be used as prefix for certain conversion characters

h    for short integers

l    for long integers or double

L    for long double

## 4.15 What happens when an input data item contain

The consecutive non-whitespace characters that define a data item collectively define a **field.** It is possible to limit the number of such characters by specifying a maximum field width for that data item. To do so, an unsigned integer indicating the field width is placed within the control string, between the percent sign(%) and conversion character.

(a) If input data has more characters than the specified field width extra characters will be ignored.

(b) If input data has fewer characters than the specified field width the input data will remain unchanged.

**Example:** If a and b are two integer variables and the following statement is being used to read their values:

scanf( "%3d %3d", &a, &b);

and if the input data is: 1 4

then a will be assigned 1 and b 4.

**Mohammad Abir Reza [CSE- 5TH BATCH]**
**Department of Computer  Science & Engineering**
**Comilla University**

If the input data is 123 456 then a=123 and b=456.

If the input is 1234567, then a=123 and b=456. 7 is ignored.

If the input is 123 4 56 (space inserted by a typing mistake), then a=123 and b=4. This is because the space acts as a data item separator.

## 4.16  what is the purpose of  printf()  function ?

printf() function is standard output function in C programming language. This function is defined inside header file called 'stdio.h'. printf() means print formatted. Output data can be written from the computer onto a standard output device using the library function printf(). This function can be used to output any combination of numerical values, single characters and strings. The printf() statement provides certain features that can be effectively exploited to control the alignment and spacing of print-outs on the terminals. The general form of printf() statement is:

> printf("control string", arg1, arg2, …, argn);

control string consists of three types of items:

1. Characters that will be printed on the screen as they appear.

2. Format specifications that define the output format for display of each item.

3. Escape sequence characters such as \n, \t, and \b.

The control string indicates how many arguments follow and what their types are. The arguments arg1, arg2, …, argn are the variables whose values are formatted and printed according to the specifications of the control string. The arguments should match in number, order and type with the format specification.

## 4.17 describe the purpose of commonly used conversion characters in a printf() function ?

The main purpose of commonly used conversion characters in a printf() function is these indicate what type of data we print as output.

Commonly used scanf() format codes are

| Code | Meaning |
|------|---------|
| %c | print a single character |
| %d | print a decimal integer |
| %e | print a floating point value in exponent form |
| %f | print a floating point value without exponent |
| %g | print a floating point value either e-type or f-type |
| %i | print a signed decimal integer |
| %o | print an octal integer without leading zero |
| %s | print a string |
| %u | print an unsigned decimal integer |
| %x | print a hexadecimal integer, without leading Ox |

The following letters may be used as prefix for certain conversion characters.

h        for short integers

l        for long integers or double

L        for long double

## 4.17 How does a control string in a printf() function differ from the control string in a scanf() function?

Control string in a printf() function can contain escape sequence characters such as \n, \t, \b. But control string in a scanf() function can include blanks, tabs, or newlines. But blanks, tabs or newlines are ignored. Control string in a printf() function can consists of characters that will be printed on the screen as they appear.

## 4.19  What happens when an 0utput data item contain ?

 (a) more characters than the specified field width and

(b) fewer characters than the specified field width?

 (a) If an output data items contains more characters than the specified field width, it will be printed in full, overriding the minimum specification. The number is written right-justified in the given field width.

(b) If an output data items contains more characters than the specified field width,  blanks are added to the left or the right of the output data items depending on whether the left alignment flag (-) is specified, until the minimum width is reached.

The following examples illustrate the output of the number 9876 under different formats:

| Format | Output | | | | | |
|--------|---|---|---|---|---|---|
| printf("%d",9876) | 9 | 8 | 7 | 6 | | |
| printf("%6d",9876) | | | 9 | 8 | 7 | 6 |
| printf("%2d",9876) | 9 | 8 | 7 | 6 | | |
| printf("%-6d",9876) | 9 | 8 | 7 | 6 | | |

## 4.20 How are the unrecognized characters within the control string are interpreted in

## (a) scanf function; and

## (b) printf function?

STEVE JOBS

DESIGNER OF NEW WORLD

(a) Unrecognized characters within the control string are expected to be matched by the same characters in the input data. The unrecognized characters will be read into the computer but not assigned to an identifier. Execution of the scanf function will terminate if a match is not found
Example:    scanf("%d * %d",&a,&b);
the accepted  input is 5 * 4. If we give a and b as 5 and 4 then the program will stop executing as the expected character * is not found. so a is assigned to 5 and b is assigned to 0.

(b) Unrecognized characters within the control string are printed in output in printf function.

# Chapter – 05

# Decision making and Branching

**5.1:State whether the following are true or false :**

(a)When if statements are nested , the last else gets associated with the nearest  if without an else.
    Ans: False.
(b)One if can have more than one else clause.
    Ans: False.
(c)A switch statement can always be replaced by a series of if..else statements.
    Ans: False.
(d)A switch expression can be of any type.
    Ans: False.
(e)A program stops its execution when a break statement is encountered.
    Ans: False.
(f)Each expression in the else if must test the same variable.
    Ans: True.
(g)Any expression can be used for the if expression.
    Ans: True.
(h)Each case label can have only one statement.
    Ans: True.
(i)The default case is required in the switch statement.
    Ans: True.
(j)The predicate   !( (x>=10) | (y==5) ) is equivalent to (x<10) && (y!=5).
    Ans: False

## 5.2:Fill in the blanks in the following statements:

(a)The ……….operator is true only when both the operands are true.

Ans: logical AND (&&).

(b)Multiway section can be accomplished using an else if statement or the …………… statement.

Ans: switch.

(c)The……….. statement when executed in a switch statement causes. immediate exit from the structure

Ans: break.

(d)The ternary conditional expression using the operator ?: code be easily coded using ………..statement.

Ans: if…else.

(e)The expression !(x!=y)can be replaced by the expression…………

Ans: x==y.

5.3:Find errors, if any, in each of the following segments:

(a)if((x+y=z) && (y>0) )
   Printf (" ");
Answer: Error.
Correct answer:  if((x+y==z) && (y>0) )
                printf(" ");

(b) if (code >1)
     a= b+c
     else
     a=0
   Answer:  Error.
 Correct answer: if (code >1)
              a= b+c ;
            else
              a=0 ;

(c) if(p>0) || (q <0)
     printf("Sign is negative");
   Answer:  Error.
 Correct answer:  if((p>0) || (q <0))
              printf("Sign is negative");

STEVE JOBS
DESIGNER OF NEW WORLD

**5.4:The following is a segment of a program:**

```
x=1;
y=1;
if(n>0)
    x=x+1;
y=y-1;
printf("%d  %d", x,y);
```

N.B: This statement is not associated with if condition.

**What will be the values of x and y if n assumes a value of (a) 1and (b) 0.**
**Solution:**
    **(a)2,0**
    **(b) 1,0**

**5.5:Rewrite each of the following without using compound relations:**

(a)if (grade<=59 && grade>=50)
   second=second+1;

Answer:
if (grade<=59)
   if(grade >=50)
     Second=second+1;

STEVE JOBS
DESIGNER OF NEW WORLD

(b)  if (number>100 || number<0)
     Printf("out of range");
  else
    Sum=sum+ number ;

Answer:
if (number>100)
   Printf("Out of range");
else if (number<0)
   Printf("Out of range");
else
   Sum=sum+1;

(b) if ((M1>60  &&  M2>60) || T>200)
      printf("Admitted\n);
    else
      printf("Not admitted\n");


answer :

```
if (M1>60)
  {
    if (M2 >60)
        Printf("Admitted\n");
     else if (T>200)
          Printf("Admitted\n);
  }
  else if (T>200)
          Printf("Admitted\n);
  else
       Printf("Not admitted\n);
```

**5.6:Assuming x=10 ,state whether the following logical expressions are true or false:**
    (a)x==10 && x>10 && !x         Ans:False.
    (b)x==10 || x> 10 && !x          Ans:True.
    (c)x==10 && x>10 ||!x           Ans:False.
    (d)x==10 ||x>10 || !x           Ans:True.

STEVE JOBS

DESIGNER OF NEW WORLD

**5.7:Find errors,if any, in the following switch related statements. Assume that the variables x and y are of int type and x=1 and y=2.**
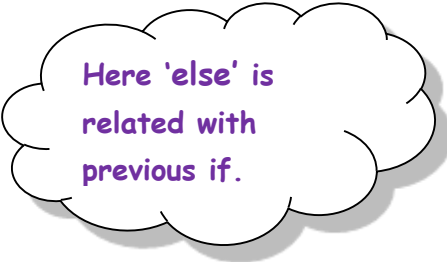    (a)switch(y);
        Answer : Error. Correct answer: switch(y)

    (b)case 10;
        Answer : Error.Correct answer: case 10:

    (c)switch(x+y)
      Answer : No  error.

*Semicolon (;) after break.*

    (d)switch(x)  {Case  2:  y= x+y; break};
        Answer : Error.Correct answer: switch(x)  {Case  2:  y= x+y; break;}

**5.8:Simplify the following compound logical expressions:**

(a)!(x<=10)

Answer:(x>10)

(b)!(x==10)||!((y==5)||(z<0))

Answer: x!=10|| y!=5 && z>=0

(c)!((x+y==z)&&!(z>5))

Answer: !(x+y==z) || (z>5) = x+y!=z || z>5

(d)!((x<=5)&&(y==10)&&(z<5))

Answer: x>5 || y!=10 || z>=5

**5.9:Assuming that x=5, y=0,and z=1 initially ,what will be their values after executing the following code segments?**

(a)if(x && y)
   x=10;
   else
   y=10;
Output: x= 5, y= 10 , z= 1

(b)if(x|| y ||z)
   y=10;
   else
   z=0;
Output: x= 5, y= 10 , z= 1

(c)if(x)
    if(y)
     z=10;
    else
      z=0;
Output: x= 5, y=0 , z=0

(d)if(x ==0 || x && y)
     if(!y)
       z=0;
     else
       y=1;
Output: x= 5, y=0 , z=1

Here 'else' is related with previous if.

---

**5.10:Assuming that** *x=2,y=1* **and** *z=0* **initially ,what will be their values after executing the following code segments?**
**(a)**
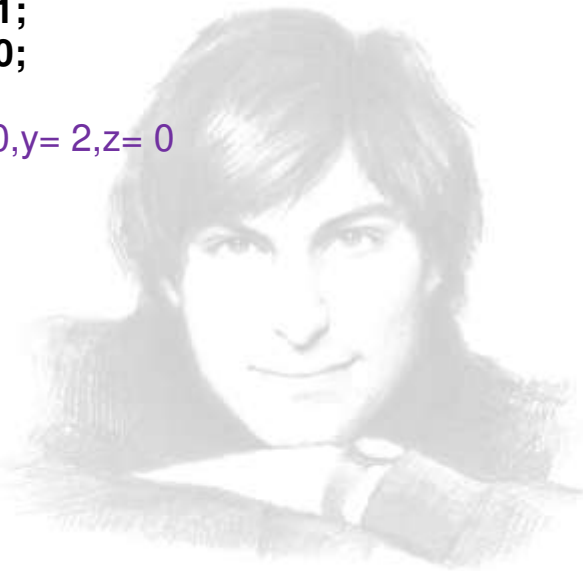
```
switch(x)
{
    case 2:
        x=1;
        y=x+1;
    case 1:
        x=0;
        break;
    default:
        x=1;
        y=0;
}
```

Output: x=0,y= 2,z= 0


**(b)**

```
switch(y)
{
case 0:
    x=0;
    y=0;
case 2:
    x=2;
    z=2;
default:
    x=1;
    y=2;
}
```

Output:  x=1 ,y= 2, z=  0

**5.11:Find the error ,if any,in the following statements**:

(a)if (x>=10)   then
        printf(" \n ")            Error
(b)  if x>=10
        Printf(" ok " );          Error
(c)  if (x=10)
         printf("GOOD");   No  error
(d)  if (x=<10)
        Printf(" Welcome ") ;   Error

**AFTER  CORRECTION**

(a)if (x>=10)
    printf(" \n ");
(b)  if ( x>=10 )
         Printf(" ok " );
(c)  if (x=10)[no error]
        printf("GOOD");
(d) if (x   <=10)
        Printf("Welcome") ;

**5.12:What is the output of the following program?**

```
main()
{
   int  m=5;
if(m<3) printf("%d", m+1);
else if (m<5) printf("%d", m+2);
else if (m<7) printf("%d", m+3);
else printf("%d", m+4);
getch();
}
```

**Output:**      **8**

**5.13:What is the output of the following program?**

```
main ()
{
   int  m=1;
   if( m==1)
      {
        printf ("Delhi");
        if(m==2)
        printf("Chennai");
     else
        printf("Banglore");
    }
   else  ;
   Printf("END");
   getch();
}
```

**Output:   Delhi Banglore END**

STEVE JOBS

DESIGNER OF NEW WORLD

Be careful- if there is
semicolon (;) after else, it
takes as a blank statement. So
here 'printf("END"); ' statement
is not working with 'else'.

**5.14:What is the output of the following program?**
**Program:**

```
main()
{
int  m;
for(m=1; m<5; m++)
printf("%d\n",(m%2) ? m : m*2);
getch();
}
```

**Output:**    1
                4
                3
                8

**5.15:What is the output of following program?**

```
main()
{
int  m,n,p;
for(m=0; m<3;m++)
for(n=0;n<3;n++)
for(p=0;p<3;p++)
if(m +n +p==2)
goto print;
print:
printf("%d %d %d", m, n, p);
getch();
}
```

If there is 'goto' statement, it comes out from the loop.

**Output:  0   0   2**

**5.16:What will be the value of x when the following segment is executed?**

```
int  x=10,y=15;
x= (x<y)? (y+x)  : (y-x) ;
```

**Solution:**
  The value of x after execution is :       **25**

**5.17:What will be the output when the following segment is executed?**

```
int  x=0;
if(x>=0)
if(x>0)
    printf("Number is positive");
else
    printf("Number is negative");
```
**Output:**
**Number is negative**


**5.18: What will be the output when the following segment is executed?**
**Program:**

```
char ch = 'a'
switch(ch)
{
        case 'a':
                printf("A");
        case 'b':
                printf("B");
        case 'c':
                printf("C");
}
```
**Output:  ABC**

**5.19:What will be the output of the following segment when executed?**
**Program:**

```
main()
{
 int  x=10,y=20;
if(( x<y)|| (x+5)>10)
    printf("%d",x);
else
    printf("%d",y);
getch();
}
```
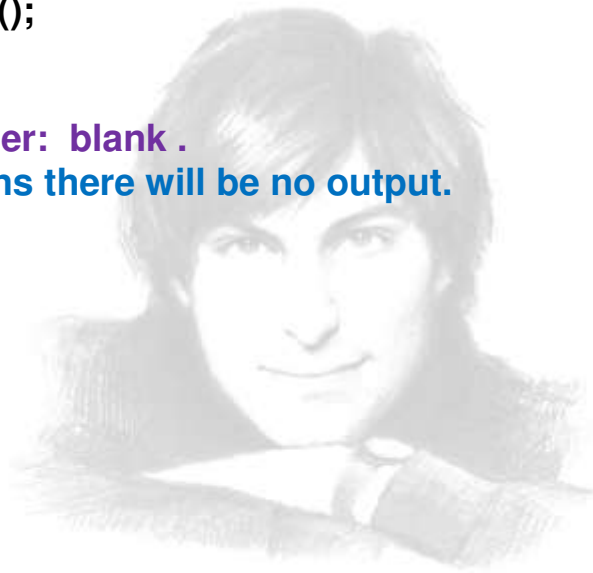**Output:  10**

**5.20:What will be the output of the following segment when executed?**

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

**Answer:  blank .**
**That means there will be no output.**

# Chapter - 06
# Decision Making and Looping

## 6.1 State whether the following statements are true or false:

(a) The do… while statement first executes the loop body and then evaluate the loop     control expression.
**Ans: True.**

 (b) In a preset loop, if the body is executed n terms,  the test expression is executed n+1 times.
 **Ans: True.**

(c) The number of times a control variable is updated always equals the number of loop iterations.
**Ans: True.**

 (d) Both the preset loops include initialization within the statement.
 **Ans: True.**

(e) In a for loop expression, the starting value of the control variable must be less than its ending value.
 **Ans: True.**

(f) The initialization, test condition and increment parts may be missing in a for statement.
 **Ans: False.**

 (g) While loops can be used to replace for loops without any change in the body of          the   loop.
**Ans: False.**

 (h) An exit control loop is executed a minimum of a one line.
 **Ans: False.**

(i) The use of continue statement considered as unstructured programming.
 **Ans: True.**
(j) The three loop expressions used in a for loop header must be separated by commas .
**Ans: True .**

## 6.2: Fill in the blanks in the following statements.

(a) In an exit controlled loop, if body is executed n times, test condition is evaluated _____ times.

**Answer: (n-1)**

(b) The _____ statements is use to skip a part of the statements in a loop.

**Answer: continue.**

(c) A for loop with the no test condition is known as _____ loop.

**Answer: infinite**

(d) The sentinel controlled loop is also; known as_____ loop.

**Answer: indefinite repetition.**

(e)In a counter controlled loop, variable known as_____ is used to count the loop operation.

**Answer: definite repetition.**

## 6.3 Can we change the value of the control variable in for statements? If yes, explain its consequences.

No, we cannot change the control variable in both the for statement and the body of the loop. It is a logic error. If we change the value of the control variable in for statement the number of iterations will be changed.

[Source : 'Just' remember part of the chapter]

## 6.4 What is a null statement? Explain a typical use of it.

➢ If we place semicolon after 'for statement' the semicolon will be considered as null statement & the compiler will produce some nonsense. It is not a syntax error, it is a logical error.
➢ If we want infinite output we can use null statement.

## 6.5 Use of goto should be avoided. Explain a typical example where we find the application of goto becomes necessary

➢ It is a good practice to avoid using goto. There are many reasons for this. When goto is used, many compilers generate a less efficient code. In addition, using many of them makes a program logic complicated and renders the program unreadable.

But since a goto statement can transfer the control to any place in a program, it is    useful to provide branching within a loop. Another important use of goto is to exit from a    deeply nested loops when an error occurs.

## 6.6 How would you decide the use of one of the three loops in C for a given problem?

Given a problem, the programmer's first concern is to decide the type of loop structure to be used. To choose one of the three loop supported by C, we may use the following strategy:

1. Analyse the problem and see whether it required a pre-test or post-test loop.

2. If it requires a post-test loop, then we can use only one loop, do while.

3. If it requires a pre-test loop, then we have two choices: for and while.

4. Decided whether the loop termination requires counter-based control or sentinel-based control.

5. Use for loop it the counter-based control is necessary.

6. Use while loop if the sentinel-based control is required.

7. Note that both the counter-controlled and sentinel-controlled loops can be implemented by all the three control structures.

## 6.7 How can we use for loops when the number of iterations are not known?

It does not matter if we know the number of iterations or not, the for loop is the same.

## 6.8  Explain the operation of each of the following for loops.

(a)for (n=1;n!=10;n+=2)
   sum=sum +n;

Answer :The loop will compute the sum of n odd numbers.

(b)for(n=5;n<=m ; n-=1)
   sum+=n;

Answer: The continue until n<=m  becomes false, where    n initializes from   5   and   decrements by 1.

(c) for(n=1;n<=5 ; )
   sum+=n;

Answer: The loop repeats  infinity times, as there is no increment or decrement.

(d) for(n=1;  ;n+=1)
   sum+=n;
 Answer: The loop repeats  infinity times , as there is no condition.

(e)for(n=1;n<5;n++)
   n=n-1;

Answer: The loop repeats  infinity times.

**Mohammad Abir Reza [CSE- 5TH BATCH]**
**Department of Computer  Science & Engineering**
**Comilla University**

## 6.9: What would be the output of each of the following code segments?

```
(a)count=5;
      while(count-- >0)
   printf(count);
```

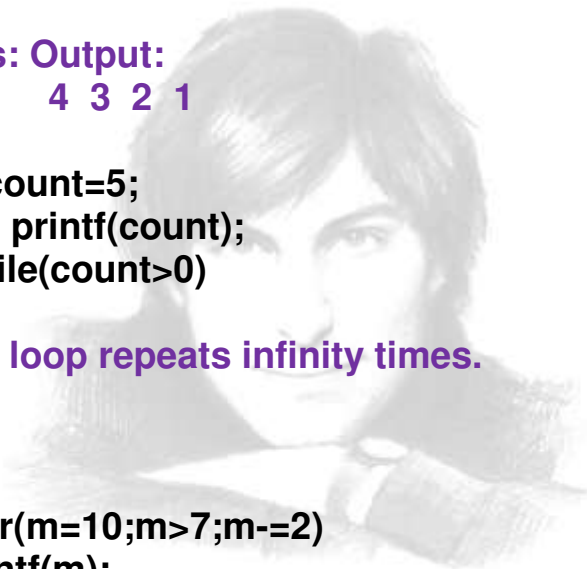Output:    4  3  2  1  0

```
(b)count=5;
      while(-- count>0)
      Printf(count);
```

Ans: Output:
        4  3  2  1

```
(c)  count=5;
  do  printf(count);
  while(count>0)
```

Ans: loop repeats infinity times.

STEVE JOBS

DESIGNER OF NEW WORLD

```
(d)for(m=10;m>7;m-=2)
  printf(m);
```
output;
10  8

## 6.10 Compare, in terms of their functions, the following pairs of statements:

**(a) while and do…while (b) while and for (c) break and goto (d) break and continue (e) continue and goto.**

**<u>while:</u>**

The simplest of all the looping structures in C is the while statement. The basic format of the while statement is

while (test condition)

{

        body of the loop

}

The while is an entry-controlled loop statement. The test-condition is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test-condition is once again evaluated and if it is true, the body is executed once again. This process of repeated execution of the body continues until the test-condition finally becomes false and the control is transferred out of the loop. On exit, the program continues with the statement immediately after the body of the loop. The body of the loop may have one or more statements. The braces are needed only if the body contains two or more statements. However, it is a good practice to use braces even if the body has only one statement.

**<u>do…while:</u>** The basic format for the do statement is:

do

{

        body of the loop

}

while (test-condition);

On reaching the do statement, the program proceeds to evaluate the body of the loop first. At the end of the loop, the test-condition in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.

Since the test-condition is evaluated at the bottom of the loop, the do…while construct provides an exit-controlled loop and therefore the body of the loop is always executed at least once. A goto is often used at the end of a program to direct the control to go to the input statement, to read further data.

## for:

The for loop is another entry-controlled loop that provides a more concise loop control structure. The general form of the for loop is

for (initialization ; test-condition ; increment)

{

      body of the loop

}

The execution of the for statement is as follows:

1. Initialization of the control variables is done first, using assignment statements such as i=1 and count=0. The variables i and count are known as loop-control variables.

2. The value of the control variable is tested using the test-condition. The test-condition is a relational expression, such as i < 10 that determines when the loop will exit. If the condition is true, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.

3. When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. Now, the control variable is incremented using an assignment statement such as i=i+1 and the new value of the control variable is again tested to see whether if satisfies the loop condition. If the condition is satisfied, the body of the loop is again executed. This process continues till value of the control variable fails to satisfy the test-condition.

One of the important points about the for loop is that all the three actions, namely initialization, testing, and incrementing, are placed in the for statement itself, thus making them visible to the programmers and users, in one place

| for | while | do |
|-----|-------|-----|
| for(n=1;n<10;++n)<br>{<br>     .........<br>}<br><br>    n=n+1;<br>    } | n=1;<br>while(n<=10)<br>    {<br>    .......<br>    n=n+1;<br><br>    } | n=1;<br>do<br>    {<br>    ........<br><br><br>    while(n<=10); |

fig: Comparison of the three loops

## Break:

When a break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the break would only exit from the loop containing it. That is, the break will exit only a single loop.
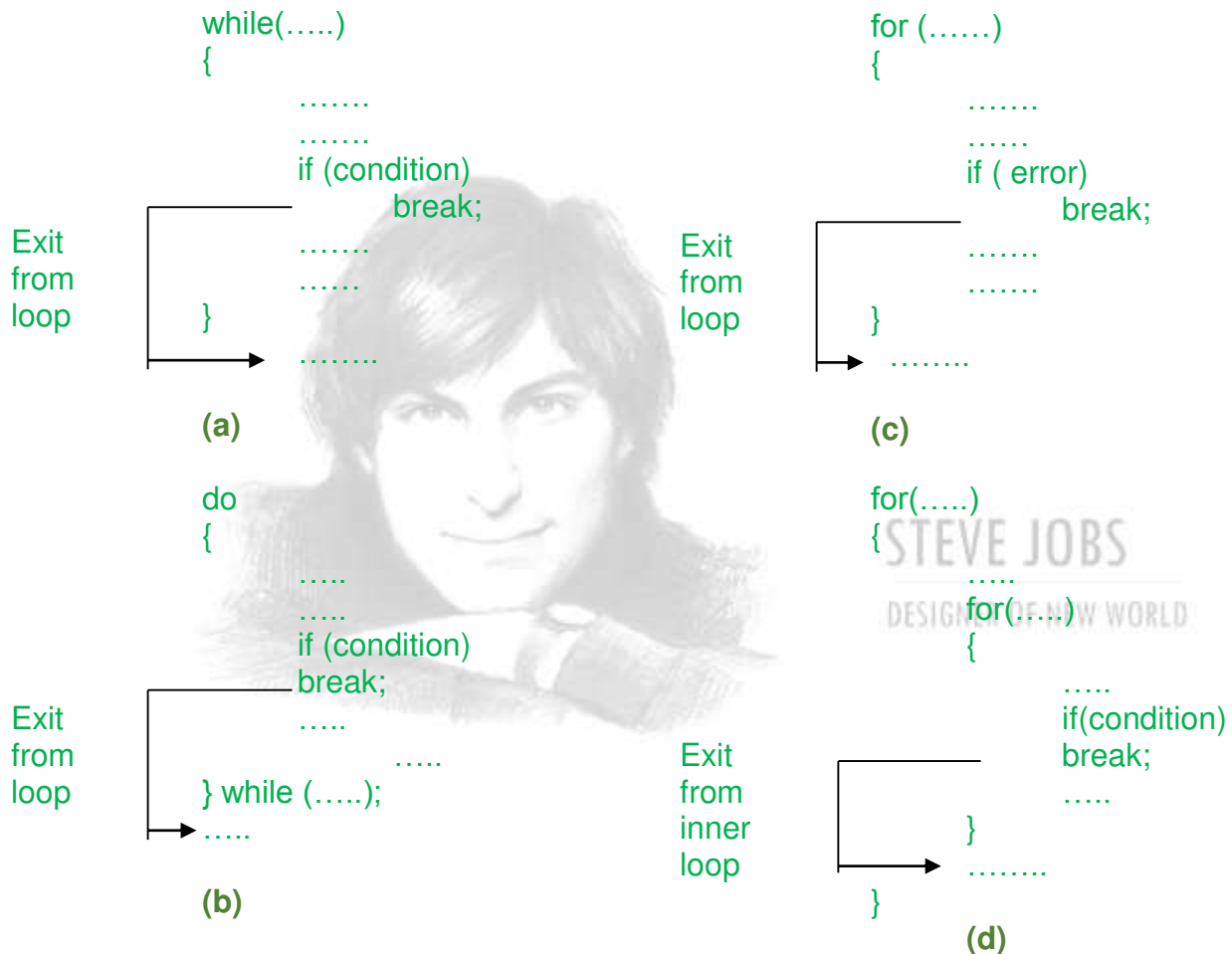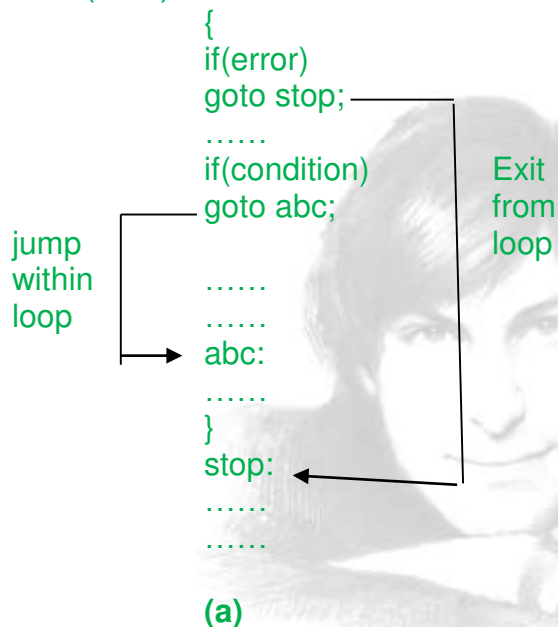
```
while(…..)                                    for (……)
{                                             {
        …….                                         …….
        …….                                         ……
        if (condition)                              if ( error)
                break;                                      break;
Exit    …….                           Exit           …….
from    ……                            from           …….
loop    }                             loop           }
        ………                                          ………

        (a)                                           (c)


do                                            for(…..)
{                                             {
        ….                                          …..
        ….                                          for(…..)
        if (condition)                              {
        break;                                              …..
Exit    ….                                                  if(condition)
from                …..                       Exit          break;
loop    } while (…..);                         from          …..
        …..                                   inner         }
                                              loop          ………
        (b)                                   }

                                                      (d)
```

Fig: Exiting a loop with break statement

MOHAMMAD ABIR REZA  (CSE-5TH BATCH)
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COMILLA UNIVERSITY

## Goto:

C supports the goto statement to branch unconditionally from one point to another in the program. Since a goto statement can transfer the control to any place in a program, it is useful to provide branching within a loop. Another important use of goto is to exit from deeply nested loops when an error occurs. It can be used to transfer the control out of a loop (or nested loops) when certain peculiar conditions are encountered.

```
while(……)
        {
        if(error)
        goto stop;
        ……
        if(condition)           Exit
        goto abc;               from
                                loop
jump
within   ……
loop     ……
         abc:
         ……
         }
         stop:
         ……
         ……
```

**(a)**

STEVE JOBS

DESIGNER OF NEW WORLD

## Continue:

During the loop operations, it may be necessary to skip a part of the body of the loop under certain conditions. C supports a statement called the continue statement. The continue, as the name implies, causes the loop to be continued with the next iteration after skipping any statements in between. The continue statement tells the compiler, "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION". The format of the continue statement is simply.
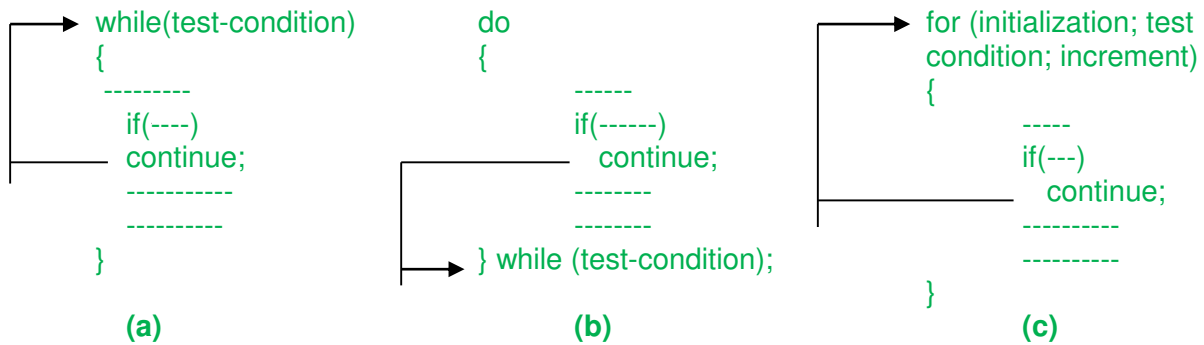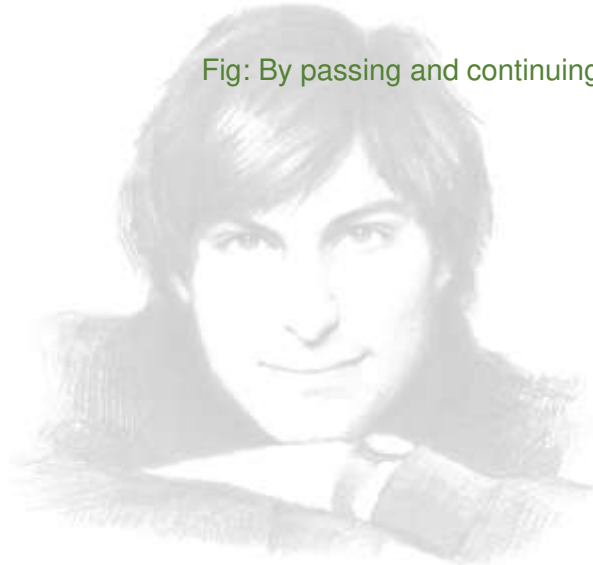
```
        continue;
```

```
while(test-condition)          do                              for (initialization; test
{                              {                               condition; increment)
  ---------                      ------                        {
    if(----)                       if(------)                      -----
    continue;                         continue;                    if(---)
  -----------                      --------                           continue;
  ----------                       --------                        ---------
}                              } while (test-condition);            ----------
                                                                }
   (a)                              (b)                             (c)
```

Fig: By passing and continuing in loops

STEVE JOBS

DESIGNER OF NEW WORLD

## 6.11:Analyse each of the program segments that follow and determine how many times the body of each loop will be executed.

(a) x=5;
    y=50;
    while(x<=y)
    {
      x=y/x;
      …………………..
      …………………..
    }
 Answer:  Infinity  times.

(b) m=1;
    do
    {
      …………………..
      …………………..
      m+=2;
    }
    while(m<10)
Answer : 5 times.

(c) int  i;
    for(i=0;i<=5;i=i+2/3)
    {
      …………………..
      …………………..
    }
Answer : Infinity times.

Here in 6.11(c) , value of  2/3 is 0(zero),  according to the rules of integer arithmetic expression.

If  a  and  b  are integer value  and  a<b  then, a/b=0(zero)

In this  loop  ' i '  increments by  0(zero), so the condition is true for  infinity times

(d) int m=10;
    int n=7;
    while(m%n>=0)
    {
      …………….
      m+=1;
      n+=2;
      …………….
    }
Answer : Four times

**6.12:  Find errors, if any, in each of the following looping segments. Assume that all the variables have been declared and assigned values.**

```
(a)while(count!=10);
{
count=1;
sum=sum +x;
count=count+1;
}
```
   Answer : Error

```
(b)
 name=0;
do
{
name=name+1;
printf("My name is Jhon\n");}
while(name=1)
```

Answer:  Error
Correct Answer:
```
 name=0;
do
{
name=name+1;
printf("my name is jhon\n");
}
 while(name==1);
```

```
(c)
  do ;
total=total+value;
 scanf("%f",&value);
while(value!=999) ;
```

Answer: Error

In the problem of 6.12(a),  there should not be any use of semicolon(;)  after <u>while</u> .

But at the case of <u>do ……. …While</u>  loop there must be a semicolon after <u>while</u>  , problem shown in 6.12(b) .

STEVE JOBS
DESIGNER OF NEW WORLD

There are  two  error in 6.12(c)-

- There should not any semicolon(;)  after  do statement.
- Brace  " { }" is needed between do and while, if there are more than one statement .

**Mohammad Abir Reza(CSE – 5th batch)**
**Department of computer science & engineering,**
**Comilla University**

Correct Answer:

```
 do
 {
 total=total+value;
 scanf("%f",&value);
 }
 while(value!=999);
```

**(d)for(x=1,x>10;x=x+1)**
**{**
**-------**
**}**
**Answer:  Error.**

6.12 (d), After assignment there should not be comma(,)  instead of semicolon(;)

**(e) m=1;**
**n=0;**
**for(;m+n<10;++n);**
**  printf("Hello\n");**

**  m=m+10**

**Answer:   Error**

```
m=1;
n=0;
for(;m+n<10;++n)
     printf("Hello\n");
      m=m+10 ;
```

**(f)for(p=10;p>0;)**
**p=p-1;**
**printf("%f",p);**
**Answer: No  Error**

Here :  printf("%f",p);  statement is not under control of for loop.

**6.13:Write a for statement to print each of the following sequence of integers:**

**(a) 1,2,4,8,16,32**

Answer: for(i=1;i<=32;i=i*2)
 printf("%d",i);

**(b) 1,3,9,27,81,243**
Answer: for(i=1;i<=243;i=i*3)
printf("%d",i);

**(c) -4,-2,0,4**
Answer:for(i=  - 4;i<=4;i=i+2)
printf("%d",i);

**(d) -10,-12,-14,-18,-26,-42**

```
#include <stdio.h>
int main()
{
   int a,m;
   m=1;
   for (a=-10;a>=-10;)
   {
       printf("%d",a);
       for(a=-12; a>=-42;a=a-m)
       {
        m=m*2;
         printf("%d",a);
       }
   }
}
```
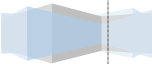
## 6.14: Change the following for loops to while loops :

```
(a)for(m=1;m<10;m=m+1)
printf(m);
```

Answer :
```
 m=1;
while(m<10)
{
m++;
printf(m);
}
```

```
(b)for(;scanf("%d",&m)!=-1;)
printf(m);
```

Answer:
```
while(scanf("%d",&m)!=-1)
printf(m);
```

## 6.15 Change the for loops in Exercise 6.14 to do loops .

(a)

```
main()
{
    int m=1;
    do
    {
        printf("%d",m);
        m++;
    }
    while(m<10);
}
```
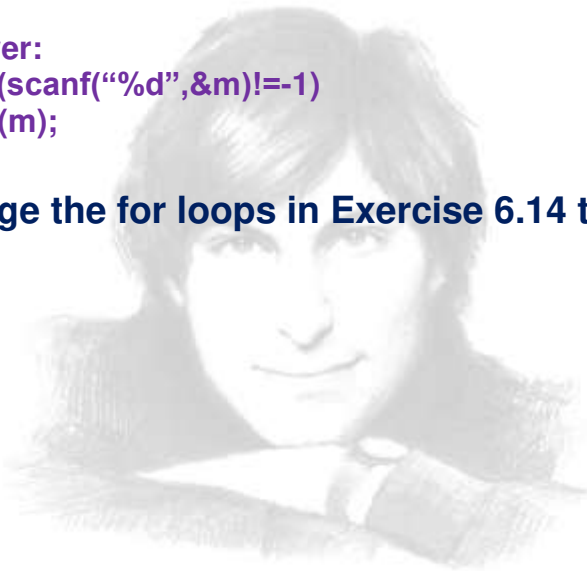
## 6.16  What is the output of following code?

```
int m=100,n=0;
while(n==0)
{

if(m<10)
break;
m=m-10;
}
```

**Output:   0**
**[N.B: The value of m will be decreased by 10 from 90 to 0. So ultimately the value of m will be 0. ]**

## 6.17: What is output of the following code?

```
int m=0;
do
{
if(m>10)
   continue;
   m=m+10;
}
while(m<50);
 printf("%d",m);
```

STEVE JOBS

DESIGNER OF NEW WORLD

**Answer :  No Output**

## 6.18: What is the output of the following code?

```
int n=0,m=1;
do
{
printf(m);
   m++;
}
while(m<=n);
```
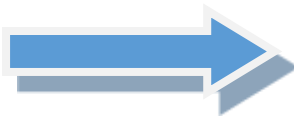
*There is a single mistake   inside the  printf function in the text book.*

*If  the function is -      printf("m");   then the output is   m  .*

*Else if the function is -   printf("%d", m);   then the output is  1  .*

**Mohammad Abir Reza(CSE – 5th batch)**
**Department of computer science & engineering,**
**Comilla University**

## 6.19: What is the output of the following code?

```
int n=0,m;
for(m=1;m<=n+1;m++)
        print(m);
```

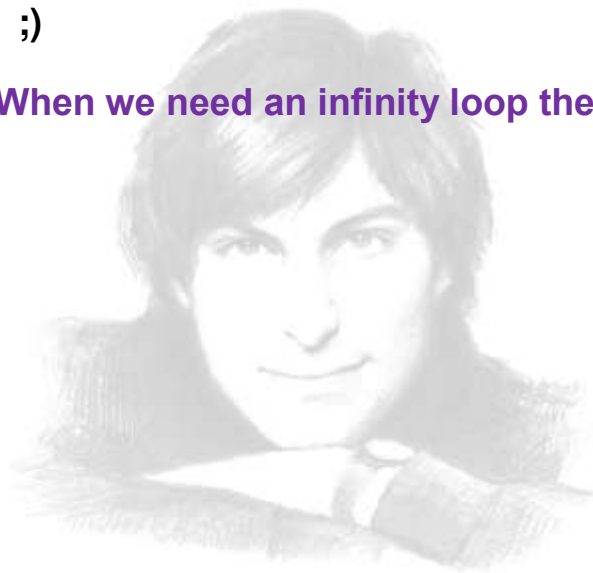*Here the same problem, printf function can't be declared in such a way.*

*If the function is -  printf("m"); then the output is  **m** .*

*Else if the function is -   printf("%d", m);  then the output is  **1***

## 6.20: When do we use the following statement?

```
for(;  ;)
```

**Answer : When we need an infinity loop the statement ' for(; ;) ' can be used.**

STEVE JOBS

DESIGNER OF NEW WORLD

# Chapter - 07
# ARRAYS

## 7.1: State whether the following statements are true or false.

(a) The type of all elements in an array must be the same.
Answer: True.
(b) When an array is declared, C automatically initializes its elements to zero.
Answer: True.
(c) An expression that evaluates to an integral value may be used as a subscript.
Answer: True.
(d) Accessing an array outside its range is a compile time error.
Answer: True.
(e) A char type variable can not be used as a subscript in an array.
Answer: True.
(f) An unsigned long int type can be used as a subscript in an array.
Answer: True.
(g) In C, by default, the first subscript is zero.
Answer: True.
(h) When initializing a multidimensional array, not specifying all its dimensions is an error.
Answer: True.
(i) When we use expression as a subscript, its result should be always greater than zero.
Answer: True.
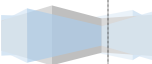(j) In C, we can use a maximum of 4 dimensions of an array.
Answer: False.
(k) In declaring an array, the array size can be constant or variable or an expression.
Answer: False.
(l) The declaration int x[2]={1,2,3};is illegal.
Answer: True.

## 7.2: Fill in the blanks in the following statements.

(a) The variable used as a subscript in an array is popularly known as…… variable.

Answer:   index.

(b) An array can be initialized either at compile time or at ………..

 Answer:   run time.

(c) An array created using malloc function at run time is referred to as ......array.

Answer:   pointer  variable.

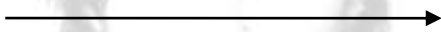(d) An array that uses more than two subscripts is referred to as …….. array.

Answer:   multidimensional

(e) ……… is the process of arranging the elements of an array in order.

Answer:   sorting.


## 7.3: Identify errors, if any, in each of the following array declaration statements, assuming that ROW and COLUMN are declared as symbolic constants :
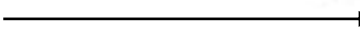
(a)int score (100);
Answer: Error. ⟶          int score [100];


(b) float values [10,15];
Answer: Error . ⟶           float values [10][15];


(c) float average [ROW] ,  [COLUMN];
Answer:Error  . ⟶ float  average  [ROW] [COLUMN];


(d) char name [15];
Answer: Correct.

> Array size missing in the declaration

(e) int sum [];
Answer:Error

(f) double salary [I +ROW]
Answer: Incorrect.

> If there is white space between 'array' and 'x' ,the compiler will show error.

(g) long int number [ROW]
Answer: Incorrect.

(h) int array   x[COLUMN];

Answer: Incorrect.(no white space between array &  x)
## 7.4: Identify errors, if any, in each of the following initialization statements.
(a) int number []={0,0,0,0,0};
**Answer: Correct.**

(b) float  item [3] [2] ={0,1,2,3,4,5};
**Answer: Correct.**

**(c) char word []={'A','R','R','A','Y'};**

**Answer: Correct .**

int m[2][4]={{0,0,0,0},{1,1,1,1}};

or,

int m[2][4]={0,0,0,0,1,1,1,1};

(d) int m[2,4] ={(0,0,0,0)(1,1,1,1)};
**Answer: Error.**

**(e) float result [10] =0;**
**Answer: Error.**

Correct answer:

float result[10]={0};

STEVE JOBS
DESIGNER OF NEW WORLD

## 7.5: Assume that the arrays A and B are declared as follows:
    int A [5] [4];
    float B [4];

**Find the errors (if any) in the following program segments.**

 (a) for (i=0;i<=5;i++)
        for(j=1;j<=4;j++)
           A [i] [j] =0;
  Answer: No  error.


(b) for (i=1;i<4;i++)
        scanf("%f",B[i]);
  Answer: Error
  Correction: for (i=1; i<=4; i++)
                scanf ("%f", &B[i]);

---

(c) for(i=0 ; i<=4 ; i++)
    B[i] =B [i] + i ;
   Answer: No Error.

 (d) for (i=4;i>=4;i--)
     for (j=0;j<4;j++)
         A [i] [j] =B [j] +1.0;
Answer: No error.

## 7.6: write a *for* loop statement that initializes all the diagonal elements of an array to one and other to zero as shown below. assume 5 rows and 5 columns.

| 1 | 0 | 0 | 0 | 0 | .............. | 0 |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | .............. | 0 |
| 0 | 0 | 1 | 0 | 0 | .............. | 0 |
| - | - | - | - | - | | - |
| - | - | - | - | - | | - |
| - | - | - | - | - | | - |
| - | - | - | - | - | | - |
| - | - | - | - | - | | - |
| - | - | - | - | - | | - |
| 0 | 0 | 0 | 0 | - | .............. | 1 |

**Answer: for(i=0;i<5;i++)**
      **for(j=0;j<5;j++)**
    **{**
     **if(i==j)**
     **printf("1");**
   **else**
       **printf("0");**
    **}**

## 7.7: We want to declare a two-dimensional integer type array called matrix for 3 rows and 5 columns. Which for the following declarations are correct?

a) int matrix [3],[5];
   Answer: Incorrect

b) int matrix [5] [3];
   Answer: Incorrect

c) int matrix [1+2] [2+3];
Answer: Correct

d) int matrix [3,5];
   Answer: Incorrect

e) int matrix [3] [5];Answer: Correct

(b)  It is a logical error.

(d)Compile time error.

## 7.8: Which of the following initialization statements are correct?

a) char str1[4]="GOOD";
Answer: Incorrect

b) char str2[ ]="C";
Answer: Correct

c) char str3[5]="MOON";
Answer: Correct

d) char str4[ ]={'S','U','N'};
Answer: This is correct.

e) char str5[10]="Sun";
Answer: Correct

Correct answer: char str1[5]="GOOD";

Cause- character array ends with null character ('\0'). So, in this array it allocates total five space in the memory.

STEVE JOBS

DESIGNER OF NEW WORLD

## 7.9: What is a data structure? Whey is an array called a data structure?

A **data structure** is a particular way of storing and organizing data in a computer so that it can be used efficiently.

An **array** is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key. An array is stored so that the position of each element can be computed from its index .

For example, an array of 10 integer variables, with indices 0 through 9, may be stored as 10 words at memory addresses 2000, 2004, 2008, … 2036, so that the element with index i has the address 2000 + 4 × i.

**Mohammad Abir Reza(CSE – 5th batch)**
**Department of computer science & engineering,**
**Comilla University**

## 7.10: What is a dynamic array? How is it created? Give a typical example of a dynamic array?

**Dynamic array:** The process of dynamic memory allocation and the arrays created at run time are called dynamic array.

**How is it created:** Dynamic arrays are created using pointer variables and memory management functions Malloc , calloc  and  realloc .

These functions are included in the header file <stdlib.h> .

**Typical example of use of a dynamic array:** The concept of dynamic arrays is used in creating and manipulating data structures such as linked lists, stacks and queues.

## 7.11: What is the error in the following program?

**Answer: Array size missing in the declaration.**

**After correction:**
```
main ()
 {
        int x;
        float y [10];
        ..........
 }
```

STEVE JOBS

DESIGNER OF NEW WORLD

## 7.12: What happens when an array with specified size is assigned
### a) with values fewer than the specified size; and
### b) with values more than the specified size.

(a )  If the size of an array is specified and provided fewer initialization elements than the total length of the array , the remaining elements will be taken as zero.

(b)  If we have more assigned value than the specified size ,the compiler will produce an error.

## 7.13: Discuss how initial values can be assigned to a multidimensional array.

Multi-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces. We can initialize a three dimensional array at the time as we initialize any other variable or array.

For example,

```
int arr[3][3][3]=
    {
        {
        {11, 12, 13},
        {14, 15, 16},
        {17, 18, 19}
        },
        {
        {21, 22, 23},
        {24, 25, 26},
        {27, 28, 29}
        },
        {
        {31, 32, 33},
        {34, 35, 36},
        {37, 38, 39}
        },
    };
```

STEVE JOBS

DESIGNER OF NEW WORLD

So in the above example we have declared multidimensional array and named this integer array as "arr" which can hold 3x3x3 (27 integers) elements. We have also initialized multidimensional array with some integer values.
For example, consider 3D array as a collection of tables, to access or store any element in a 3D array we need to know first table number then row number and lastly column number. So applying above logic, the value 25 located in table no 1 row no 1 and column no 1, hence the address is arr[1][1][1].

.
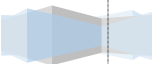
## 7.14: What is the output of the following program?

```
main ()
    {
        int m [] = {1,2,3,4,5};
        int x,y=0;
        for (x=0; x<5; x++ )
                y=y+m [x];
        printf("%d", y);
    }
```

Output: 15

## 7.15: What is the output of the following program?

```
main ()
{
        char string [ ]= "HELLO WORLD";
        int m;
        for (m=0; string [m] !='\0';m++)
                if ((m%2)==0 )
                        printf ("%c", string [m]  );
}
```
Output:   HLOWRD

STEVE JOBS

DESIGNER OF NEW WORLD

# Chapter – 08

# Character Arrays and Strings

## 8.1: State whether the following statements are *true* or *false.*

(a) When initializing a string variable during its declaration, we must include the null character as part of the string constant, like 'GOOD\0'.
Answer: False.

(b) The gets function automatically appends the null character at the end of the string read from the keyboard.
Answer: True.

(c) When reading a string with scanf, it automatically inserts the terminating null character.
Answer: True.

(d) String variables cannot be used with the assignment operator.
Answer: True.

(e) We cannot perform arithmetic operations on character variables.
Answer: True.

(f) We can assign a character constant or a character variable to an int type variable.
Answer: False.

(g) The function scanf cannot be used in any way to read a line of text with the white spaces.
Answer: True.

(h) The ASCII character set consists of 128 distinct characters.
Answer: False.

(i) In the ASCII collating sequence, the uppercase letters precede lowercase letters.
Answer: True.

(j) In C, it is illegal to mix character data with numeric data in arithmetic operations.
Answer: False.

(k) The function getchar skips white-space during input.
Answer: False.

(l) In C, strings cannot be initialized at run time.
Answer: False.

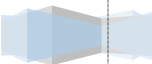(m) The input function gets has one string parameter.
Answer: True.

(n) The function call strcpy (s2, s1); copies string s2 into string s1.
Answer: False.

(o) The function call strcmp ( 'abc' , 'ABC' ); returns a positive number.
Answer: True.

## 8.2 Fill in the blanks in the following statements.

(a) We can use the conversion specification ….. in scanf to read a line of text.
Answer: code.
(b) We can initialize a string using the string manipulation function ……...
Answer: strcpy
(c) The function strncat has (three) parameters.
Answer: three
(d) To use the function atoi in a program, we must include the header file …….
Answer: <std.lib.h>
(e) The function ……. does not require any conversion specification to read a string from the keyboard.
Answer: gets.
(f) The function ……… is used to determine the length of a string.
Ans: strlen.
(g) The ……….string manipulation function determines if a character is contained in a string.
Answer: strstr.
(h) The function ………..is used to sort the strings in alphabetical order.
Answer: ASCII.
   (i) The function call strcat (s2,s1); appends …… to …….
Answer: One, another.
(j) The printf may be replaced by ……… function for printing trings.
Answer: Puts.

## 8.3: Describe the limitations of  using getchar and scanf functions for reading strings.

- By using getchar we can read only one character from the keyboard. So we can't read the full string.

- The limitation of scanf function is that it terminates its input on the first white space it finds. A whitespace  is included with blanks , tabs ,carriage returns , form feeds and new lines.

## 8.4: Character strings in C are automatically terminated by the null character.
## Explain how this feature helps in string manipulations.

We know that a string is not a data types in c, but it is considered a data structure stored in array. The string is a variable-length structure and is stored in a fixed-array. therefore, the last element of an array need not be represent at the end.
It is automatically terminate by null character.

## 8.5: Strings can be assigned values as follows:
## Compare them critically and describe situations where one is superior to others.

(a) During type declaration                char string[]={"........."};
Answer: Read a character string.

(b) Using strcpy function                strcpy(string, "......");
Answer: Copy one string to another.

(c) Reading using scanf function          scanf("%s",string);
Answer: It takes a string.

(d) Reading using gets function          gets(string);
Answer: Read a line of string.

## 8.6: Assuming the variable string contain the value "The sky is the limit.", determine

What output of the following segments will be?

(a) printf ("%s",string);

Output:

The sky is the limit.

(b) printf("%25.10s",string);

output:

The sky is

Here 15 spaces before 'The sky is'

(c) printf("%s",string[0]);

> If
>
> printf("%c",string[0]);
>
> Output : T

(d) for(i=0;string[i]!= ".";i++)
        printf("%c",string[i]);

(e) for(i=0 ; string[i]!= '\0';i++)
        printf("%d\n",string[i]);

> If there is quotation mark "." at test condition, there will error. But if there is Apostrophe '.' there will be a output & that is-
>
> the sky is the

Output:
84
104
101
32
115
107
121
32
105
115
32
116
104
101
32
108
105

109
105
116
46


(f) for(i=0;i<=strlen[string]; ;)
    {
        string[i++]=i;
        printf("%s\n",string[i]);
    }
output: Error in program

(g) printf("%c\n",string[10]+5);
output: %

(h) printf("%c\n",string[10]+'5');
output:  u


**8.7: Which of the following statements will correctly store the concatenation of strings s1 and s2 in string s3?**

    **(a) s3=strcat (s1,s2);**
    **Answer: Correct.**
    **(b) strcat(s1,s2,s3);**
    **Answer: Error.**
    **(c) strcat(s3,s2,s1);**
    **Answer: Error.**
    **(d) strcpy(s3,strcat(s1,s2));**
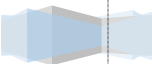    **Answer: Correct.**
    **(e) strcmp(s3,strcat(s1,s2));**
    **Answer: Error**
    **(f) strcpy(strcat(s1,s2),s3);**
    **Answer: Error.**

## 8.8: What will be the output of the following statements?

printf("%d",strcmp("push", "pull"));

### output:7

## 8.9: Assume that s1, s2 and s3 are declared as follows:

char s1[10]= "he", s2[20]= "she", s3[30], s4[30];

**What will be the output of following statements executed in sequence?**

printf("%s", strcpy(s3, s1));  ⟶  he

printf("%s", strcat(strcat(strcpy(s4,s1),"or"),s2));⟶  heorshe

printf("%d %d",strlen(s2)+strlen(s3),strlen(s4));  ⟶  5   7

## 8.10: Find errors if any, in the following code segments:

(a) char str[10]
    strncpy(str, "GOD", 3);
    printf("%s",str);
**Answer: error.**

```
char str[10];

strncpy(str,"GOD",3);

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

(b) char str[10];
    strcpy(str, "Balagurusamy");
**Answer: no error.**

(c) if strstr("balagurusamy", "guru")==0);
       printf("Substring is found");

**Answer:  error.**

if (strstr("balagurusamy", "guru")==0);
       printf("Substring is found");

(d) char s1[5], s2[10],
    gets(s1, s2);

**Answer: error.**

char s1[5],
s2[10] ;
gets(s1);

Gets function
holds one string

Mohammad Abir Reza(CSE_ 5th batch)

## 8.11: What will be the output of the following segment ?

```
char s1[]= "Kolkata";
char s2[]=  "Pune";
strcpy(s1, s2);
printf("%s",s);
```

output:  Pune

## 8.12: What will be the output of the following segment s?

```
char s1[]= "NEW DELHI" ;
char s2[]= "BANGALORE" ;
strncpy(s1, s2, 3) ;
printf("%s", s1)  ;
```

Output:  BAN  DELHI

## 8.13: What will be the output of the following code?

```
char s1[]= "Jabalpur";
char s2[]= "Jaipur";
printf(strncmp(s1, s2, 2));
```

Output:  0

## 8.14: What will be the output of the following code?

```
char s1[]= "ANIL KUMAR GUPTA"
char s2[]= "KUMAR"
printf(strstr(s1,s2));
```

output:
KUMAR GUPTA

## 8.15: Compare the working of the following functions:
### a. stcpy and strncpy:
The function strcpy copies one string to another string . It takes the form:

strcpy (string1, string2);

and assigns the contents of string 2 to string 1.

The function strncpy copies only the left-most n characters of the source string to the target string variable. . This is a three-parameter function and is invoked as follows:

strncpy(s1,s2,5);

This statement copies the first 5 characters of the source string s2 into the target string s1. Since the first 5 characters may not include the terminating null character, we have to place it explicitly in the $6^{th}$ position of s2 as shown below:

s1[6]= '\0';

Now, the string s1 contains a proper string.

### c. strcat and strncat :
**The function strcat joins two strings together**. It takes the following form:

strcat(string1,string2);

string1 and string2 are character arrays. when the function strcat is executed, string2 is appended to string1. It does so by removing the null character at the end of string1 and placing string2 from there.
**but the function strncat concanate left-most n characters of target string to source string.**

## b. strcmp and strncmp:

The strcmp function compares two strings identified by the arguments and has a value 0 if they are equal. If they are not, it has the numeric difference between the first non-matching characters in the strings. It takes the form:

strcmp(string1,string2);

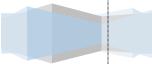string1 and string2 may be string variables or string constansts.

**The strncmp function compares the left-most n characters of two string.**

# **U**ser-Defined Functions

## 9.1: State whether the following statements are true or false.

(a) C function can return only one value under their function name.
Answer: False

(b) A function in C should have at least one argument.
Answer: True

(c) A function can be defined and placed before the main function.
Answer: False

(d) A function can be defined within the main function.
Answer: False.

(e) An user-defined function must be called at least once; otherwise a warming message will be issue.
Answer: True.

(f) Any name can be used as a function name.
Answer: False.

(g) Only a void type function can have void as its argument.
Answer: False.

(h) When variable values are passed to function, a copy of them are created in the memory.
Answer: True.

(i) Program execution always begins in the main function irrespective of location in the program.
Answer: True.

(j)  Global variable are visible in all blocks and function in the program.
Answer: False.

(k)  A function can call itself.
Answer: True

(l )  A function without a return statement is illegal.
Answer: False.

(m)   Global variable can not be declared as auto variables.
Answer: False.
(n)   A function prototype must always be placed outside the calling function.
Answer: True
(o)   The return type of a function int by default.
Answer: True.

(p)The variable names use din prototype should match those used in the function definition.
Answer: True.

(q)   In parameter passing by pointers, the formal parameters must be prefixed with the symbol * in their declarations.
Answer: True.

(r)   In parameter passing by pointers, the actual parameters in the function call may be variables or constants.
Answer: False.

(s)   In passing arrays to function, the function call must have the name of the array to be passed without brackets.
Answer: False.
(t)   In passing strings to function, the actual parameter must be name of the strings post-fixed with size in brackets.
Answer: True.

## 9.2: Fill in the blanks in the following statements.

(a)   The parameters used in a function call are called ……………
Answer: actual parameter.
(b)   A variable declared inside a function is called ……….. variable.
   Answer: local
(c)   By default………….is the return type of a function.
   Answer: int
(d)   In passing by pointers, the variable of the formal parameters
   must be prefixed with ……………operator in their declaration.
   Answer: indirection
(e)   In prototype declaration specifying .parameter……….. is optional.
   Answer: name
(f)   …………… refers to the region  where a variable is actually variable for use.
   Answer: Prototype
(g)   A function that calls itself is known as a………….. function.
   Answer: recursive

(h) If a local variable has to retain its value between calls to the function, it must be declared as ………...
Answer: void

(i) A data ……….. aids the compiler to check the matching between the actual arguments and the formal ones .
Answer: types

(j) A variable declared inside a function by default assumes …………storage class.
Answer: without

## 9.3 The main is a user-defined function. How does it differ from other user-defined function?

C function can be classified into two categories .They are library function and user defined function.
A user defined function has to be developed by the user at the time of writing a program. Main is an example of user defined functions.
Every program must have a main function to indicate where the program has to begin its execution.
No other function in the program can be called main.
It is possible to code any program utilizing only main.
Other user define function cannot call main function but main function can call other user define function.
That is why main function is different user defined functions.

## 9.4 Describe the two ways of passing parameters to function .When do you prefer to use each of them?

The technique used to pass data from one function to another is known as parameter passing. Parameter passing can be done in two ways:

1. Pass by value (also known as call by value).

2. Pass by pointers (also known as call by pointers).

In pass by value, values of actual parameters are copied to the variables in the parameter list of the called function. The called function works on the copy and not on the original values of the actual parameters. This ensures that the original data in the calling function cannot be changed accidentally.

In pass by pointers (also known as pass by address), the memory addresses of the variables rather than the copies of values are sent to the called function. In this case, the called function directly works on the data in the calling function and the changed values are available in the calling function for its use.Pass by pointers method is often used when manipulating arrays and strings. This method is also used when we require multiple values to be returned by the called function.

## 9.5:  What is prototyping? Why is it necessary?

**A function prototype is a basically a declaration for a function. Like the variables, all function in a program must be declared, before they are invoked. A function declaration consists of four parts.**

1. **Function type (return type)**
2. **Function name.**
3. **Parameter list.**
4. **Terminating semicolon.**

They are coded in the following format –
function-type   function name (parameter list);

If a function is not declared before it is used, C will assume that its details available at the time of linking. Since the prototype is not available , C will assume the return type is an integer and that the types of parameters match the formal definitions. If these assumptions are wrong , the linker will fail and we will have to change the program.

## 9.6: Distinguish between the following:

**(a)Actual and formal arguments**

- The parameters used in prototype s and definitions are called formal parameters.
- The parameter used in function call is called actual parameter.

Actual parameters must match exactly in type, order and number .
But their names , do not need to match.
Example: Suppose sum() is a function.

int sum (int x, int y) /* Here x and y are formal parameters

{

………

}

void main ()

{

    int ans;

    ans= sum (3,5); /* Here 3 and 5 are actual parameters

}

**[Note: parameter and argument is same thing]**

## (b) Global and local variables

- Local (Internal) variables are those which are declared within a particular function.
- Global (External) variables are declared outside of any function.

| Global variable | Local variable |
|---|---|
| Global variable is a variable which is declared before main function. | A variable which is declared inside the main function or other functions is called local variable. |
| A global variable can be accessed by all functions. | A local variable is isolated in its function. |
| It works with changing values. | It does not work with changing values. |
| Used only one name in a program. | Same variable names are working different values in different |

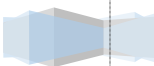## (c) Automatic and static variables:

| Static variables | Auto variables |
|---|---|
| 1. We have to specify the storage class to make a variable static. | 1. It is the default storage class. |
| 2. If it is not assigned any value then it will give 0 as output. | 2. If it is not assigned any value then it will give garbage value as output. |
| 3. It is visible to the block in which it is declared and also in the function where it will passed. | 3. It is visible to the block in which the variable is declared. |
| 4. It retains its value between different function calls. It holds its last value. | 4. It retains its value till the control remains in the block in which the variable is declared. |
| 5. Static variable should be compiles by compiler first. | 5. Auto variable will compile by the compiler after the static variable. |
| 6. A static variable maybe either an internal type or external type depending on the place of declaration. | 6. Auto variable is always internal type. |

## (d) Scope and visibility of variables:

**Scope:**

The region of a program in which a variable is available for use.

**Visibility:**

The program's ability to access a variable from the memory.

**Mohammad Abir Reza(CSE – 5th batch)**
**Department of computer science & engineering,**
**Comilla University**

**(e) & operator and * operator**

The operator & is called the address operator.
The operator * is known as indirection operator because it gives an indirect reference to a variable through its address.

## 9.7   Explain what is likely to happen when the following situations are   encountered in a program .

.

a. Actual arguments are less than the formal arguments in a function.

If the actual are less than the formals, the unmatched formal arguments will be initialized to some garbage.

b. Data type of one of the actual arguments does not match with the type of the corresponding formal argument.

When data type of one of the arguments does not match with the type of the corresponding formal argument will results in some garbage.

c. Data type of one of the argument in a prototype does not match with the type of the corresponding formal parameter in the header line.

When data type of one of the arguments in a prototype does not match with the type of corresponding formal parameter in the header line, compiler will produce an error.

d. The order of actual parameters in the function call is different from the order of formal parameters in a function where all the parameters are of the same type.
When the order of actual parameters in the function call is different from the order of formal parameters in a function where all the parameters are of the same type, the program will produce wrong output.

e. The type of expression used in return statement does not match with the type of the function.

When the type of expression used in return statement does not match with the type of the function, the return value is automatically cast to the function's type.

## 9.8 Which of the following prototype declarations are invalid? Why?

**(a) int (fun) void;**
**Answer:invalid ………… Correct answer: int fun ( void);**

**(b)double fun (void)**
**Answer:invalid ……….Correct answer: double fun (void);  (Semicolon)**

**(c) float fun (x,y,n);**
**Answer: invalid……….Correct answer: float fun (float x,float y,float n);**
**(every variable should have datatype)**

**(d) void fun (void, void);**
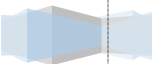**Answer: invalid………Correct answer: void fun (void);**

**(e)int fun (int a, b);**
**Answer: invalid………Correct answer: int fun (int a, int b);**
**(every variable should have datatype )**

**(f)fun (int, float, char );**
**Answer: valid**

**(g)void fun (int a, int  &b);**
**Answer: Invalid……..Correct answer: void fun (int a, int b);**

**Mohammad Abir Reza(CSE – 5th batch)**
**Department of computer science & engineering,**
**Comilla University**

## 9.9 Which of the following header lines are invalid? Why?

Header line means DEFINITION.
At the case of definition semicolon must not be used.

**(a)** float average ( float x, float y, float z);
   Answer: **invalid**……… float average (float x,float y,float z)

(b) double power ( double a, int n-1)
   Answer: **invalid**……… double power (double a, double n)
   {Note that – expression can not be used in definition}

**(c)** int product ( int m, 10)
   Answer:**Invalid**…… int product (int m, int 10)

**(d)** double minimum ( double x; double y ; )
   Answer:Invalid………. double minimum (double x, double y)

**(e)** int mul ( int x , y)
   Answer:Invalid…….. int mul (int x, int y)

**(f)** exchange ( int *a, int *b)          [no error]
   Answer:valid…………… exchange (int *a, int *b)
   [Note: If there is no return type, it means as an integer to be returned ]

**(g)** void sum ( int a, int b, int &c)
   Answer:Invalid…………… void sum (int a, int b ,int c)

## 9.10: Find errors , if any, in the following function definitions:

(a)  void  abc ( int a, int b)
{
int  c;
.............
return  (c);
}
Answer: Error.
Return with a value , in function returning void (enabled  by default  )

(b)      int  abc ( int a, int b)
        {
            ....................
        }

Answer: No Error

(c)   int  abc( int a, int b)

        {
        double c = a+b;
        return  (c);
        }
Answer :  Error.

int  abc( int a, int b)
        {
            int c = a+b;
            return  (c);
                    }

(d)     void  abc  ( void)
    {
    ..................
    ...................
    return;
    }

Answer :  No error.
(e)     int  abc ( void)
    {
    ..........................
    ..........................
    return ;
    }
Answer:  No Error.

---

**Mohammad Abir Reza(CSE – 5th batch)**
**Department of computer science & engineering,**
**Comilla University**

## 9.11 Find errors in the following function calls:

**On the occasion of FUNCTION CALL semi-colon(;) must be used**

**(a)void xyz();**
**Answer**: Error.

**(b)xyx (void);**
**A**nswer: Error

**(c)xyx (int x, int y);**
**Answer: Error**

**(d)xyzz ();**
**Answer: no error.**

**(e)xyz ()+xyz();**
**Answer: no Error.**

## 9.12 A Function to divide two flowing point numbers is as follows:

divide (float  x,    float  y)
{

return (x/y);
}

**What will be the value of the following function calls**
N.B: " At the time of function call there no need to use " data type" But here in definition  if there is no data type, it returns **integer value** "

(a)    divide (10,2)
 Answer:  5
(b)   divide (9,2)
 Answer: 4

(c)   divide (4.5,1.5).
 Answer: 3
(d)   divide (2.0,3.0)
 Answer:  0

## 9.13 What will be the effect on the above function calls if we change the header line as follow:

### (a)   int divide (int x,int y)
(a)    divide (10,2)
    Answer:  5
(b)   divide (9,2)
Answer: 4
(c)    divide (4.5,1.5).
    Answer: 3
(d)   divide (2.0,3.0)
    Answer:  0

### (b)   double divide(float x, float y)
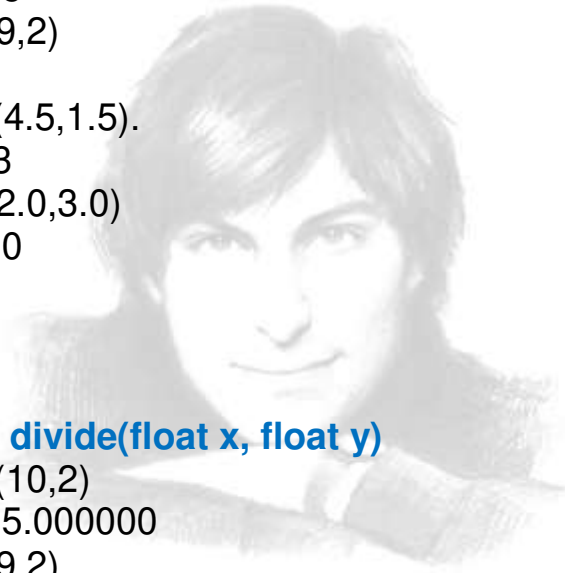(a)    divide (10,2)
    Answer:  5.000000
(b)   divide (9,2)
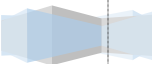Answer: 4.500000
(c)    divide (4.5,1.5).
    Answer: 3.000000
(d)   divide (2.0,3.0)
    Answer:  0.666666

## 9.14: Determine the output of the following program?

```
int prod (int m, int n);
main ( )

{
int  x=10;
int  y=20;
int  p, q;
p=prod (x,y);

q=prod (p, prod (x,Z));
printf ("%d %d\n",p,q);
}
int prod (int a,int b)
{
return  (a*b);
}
```

*If there is "Z" the program will not run, but if there is "y" it will give the answer & the output will be 200 4000*

Answer: *200 4000*

## 9.15 What will be the output of the following program?

```
void test (int *a);
main( )
{
int  x=50;
test  ( &x);
printf ("%d\n", x);
}
void  test  ( int *a) ;
{
*a=*a +50;
}
```

No semicolon after definition,

If there is a semicolon(;) there will be error in output.

Answer: 100.

## 9.15 The function test is coded as follows:

```
int  test  ( int  number)
{
int m,n=0;
while  (number)
{
m= number % 10;
if ( m% 2 )
n=n +1;
number = number/10;
}
return ( n) ;
}
```

What will be the values of x and y when the following statement are executed
int  x = test (135);
answer :  x=3

int  y=  test  (246 );
Answer : y=o

## 9.17 Enumerate the rules that apply to a function call.

A function call is a postfix expression. The operator (…) is at a very high level of precedence, therefore, when a function call is used as a part of an expression , it will be evaluated first , unless parentheses are used to change the order of precedence.

In a function call, the function name is the operand and the parameters set (..) which contains the actual parameters is the operator . The actual parameters must match the functions formal parameters in type, order and number. Multiple actual parameters must be separated by commas.
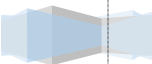
## 9.18 Summarize the rules for passing parameters to functions by Pointers.

1. The types of actual and formal argument must be same.

2. The actual arguments (in the function call) must be the addresses of variables that are local to the calling function.

3. The formal arguments in the function header must be prefixed by the indirection operator *.

4. In the prototype, the arguments must be prefixed by the symbol *.

5. To access the value of an actual argument in the called function, we must use the corresponding formal argument prefixed with the indirection operator *.

## 9.19 What are the rules that govern the passing of arrays to functions.

1. The function must be called by passing only the name of the array.

2. In the function definition, the formal parameter must be an array type; the size of the array does not need to be specified.

3. The function prototype must show that the argument is an array.

## 9.20 State the problems we are likely to encounter when we pass global variables as parameters to functions

Since all functions in a program source file can access global variables, they can be used for passing values between the functions. However, using global variable as parameters for passing value poses certain problems.

1. The values of global variables which are sent to the called function may be changed inadvertently by the called function.

2. Functions are supposed to be independent and isolated modules. This character is lost, if they use global variables.

3. It is not immediately apparent to the reader which values are being sent to the called function.

4. A function that uses global variables suffers from reusability.

STEVE JOBS

DESIGNER OF NEW WORLD

**Mohammad Abir Reza(CSE – 5th batch)**
**Department of computer science & engineering,**
**Comilla University**