# UNIT IV

## INTRODUCTION TO C

Overview of C – Constants, Variables and Data Types – Operators and Expressions– Managing Input and Output operators – Decision Making - Branching and Looping.

## 2 MARKS

1. **What are the different data types available in 'C'?**
   There are four basic data types available in 'C'.
   1. int
   2. float
   3. char
   4. double
2. **What are Keywords?**
   Keywords are certain reserved words that have standard and pre-defined meaning in 'C'. These keywords can be used only for their intended purpose.
3. **What is an Operator and Operand?**
   An operator is a symbol that specifies an operation to be performed on operands.
   **Example:** *, +, -, / are called arithmetic operators.
   The data items that operators act upon are called operands.
   **Example:** a+b; In this statement a and b are called operands.
4. **What is Ternary operators or Conditional operators?**
   Ternary operators is a conditional operator with symbols ? and :
   **Syntax:** variable = exp1 ? exp2 : exp3
   If the exp1 is true variable takes value of exp2. If the exp2 is false, variable takes the value of exp3.
5. **What are the Bitwise operators available in 'C'?**
   & - Bitwise AND
   | - Bitwise OR
   ~ - One's Complement
   >> - Right shift
   << - Left shift
   ^ - Bitwise XOR are called bit field operators
   **Example:** k=~j; where ~ take one's complement of j and the result is stored in k.
6. **What are the logical operators available in 'C'?**
   The logical operators available in 'C' are
   && - Logical AND
   || - Logical OR
   ! - Logical NOT
7. **What is the difference between Logical AND and Bitwise AND?**
   **Logical AND (&&):** Only used in conjunction with two expressions, to test more than one condition. If both the conditions are true the returns 1. If false then return 0.

**AND (&):** Only used in Bitwise manipulation. It is a unary operator.

8. **What is the difference between '=' and '==' operator?**

   Where = is an assignment operator and == is a relational operator.

   **Example:**

   while (i=5) is an infinite loop because it is a non zero value and while (i==5) is true only when i=5.

9. **What is type casting?**

   Type casting is the process of converting the value of an expression to a particular data type.

   **Example:**

   **int x,y;**

   c = (float) x/y; where a and y are defined as integers. Then the result of x/y is converted into float.

10. **What is conversion specification?**

    The conversion specifications are used to accept or display the data using the INPUT/OUTPUT statements.

11. **What is the difference between 'a' and "a"?**

    'a' is a character constant and "a" is a string.

12. **What is the difference between if and while statement?**

| if | while |
|---|---|
| (i) It is a conditional statement | (i) It is a loop control statement |
| (ii) If the condition is true, it executes some statements. | (ii) Executes the statements within the while block if the condition is true. |
| (iii) If the condition is false then it stops the execution the statements. | (iii) If the condition is false the control is transferred to the next statement of the loop. |

13. **What is the difference between while loop and do…while loop?**

    In the while loop the condition is first executed. If the condition is true then it executes the body of the loop. When the condition is false it comes of the loop. In the do…while loop first the statement is executed and then the condition is checked. The do…while loop will execute at least one time even though the condition is false at the very first time.

14. **What is a Modulo Operator?**

    '%' is modulo operator. It gives the remainder of an integer division

    **Example:**

    a=17, b=6. Then c=%b gives 5.

15. **How many bytes are occupied by the int, char, float, long int and double?**

    int - 2 Bytes
    char - 1 Byte
    float - 4 Bytes
    long int - 4 Bytes
    double - 8 Bytes

16. **What are the types of I/O statements available in 'C'?**

    There are two types of I/O statements available in 'C'.

- Formatted I/O Statements
- Unformatted I/O Statements

**17. What is the difference between ++a and a++?**

++a means do the increment before the operation (pre increment)

a++ means do the increment after the operation (post increment)

**Example:**

```
a=5;
x=a++;          /* assign x=5*/
y=a;            /*now y assigns y=6*/
x=++a;          /*assigns x=7*/
```

**18. What is a String?**

String is an array of characters.

**19. What is a global variable?**

The global variable is a variable that is declared outside of all the functions. The global variable is stored in memory, the default value is zero. Scope of this variable is available in all the functions. Life as long as the program's execution doesn't come to an end.

**20. What are the Escape Sequences present in 'C'**

| | | |
|---|---|---|
| \n | - | New Line |
| \b | - | Backspace |
| \t | - | Form feed |
| \' | - | Single quote |
| \\ | - | Backspace |
| \t | - | Tab |
| \r | - | Carriage return |
| \a | - | Alert |
| \" | - | Double quotes |

**21. Construct an infinite loop using while?**

```
while (1)
{
}
```

Here 1 is a non zero, value so the condition is always true. So it is an infinite loop.

**22. What will happen when you access the array more than its dimension?**

When you access the array more than its dimensions some garbage value is stored in the array.

**23. Write the limitations of getchar( ) and sacnf( ) functions for reading strings (JAN 2009)**

**getchar( )**

To read a single character from stdin, then getchar() is the appropriate.

**scanf( )**

scanf( ) allows to read more than just a single character at a time.

**24. What is the difference between scanf() and gets() function?**

In scanf() when there is a blank was typed, the scanf() assumes that it is an end. gets() assumes the enter key as end. That is gets() gets a new line (\n) terminated string of characters from the keyboard and replaces the '\n' with '\0'.

**25. What is a Structure?**

Structure is a group name in which dissimilar data's are grouped together.

**26. What is meant by Control String in Input/Output Statements?**

Control Statements contains the format code characters, specifies the type of data that the user accessed within the Input/Output statements.

**27. What is Union?**

Union is a group name used to define dissimilar data types. The union occupies only the maximum byte of the data type. If you declare integer and character, then the union occupies only 2 bytes, whereas structure occupies only 3 bytes.

**28. What is the output of the programs given below?**

```
main()
{
float a;
int x=6, y=4;
a=x\y;
printf("Value of a=%f", a);
}
```

**Output:**
1.

```
main()
        {
float a;
int x=6, y=4;
a=(float) x\y;
printf("Value of a=%f",a);
        }
```

**Output:**
1.500000

**29. Declare the Structure with an example?**

```
struct name
{
char name[10];
int age;
float salary;
} e1, e2;
```

**30. Declare the Union with an example?**

```
union name
{
        char name[10];
        int age;
        float salary;
} e1, e2;
```

**31. What is the output of the following program when, the name given with spaces?**

```
main()
{
        char name[50];
        printf("\n name\n");
        scanf("%s, name);
        printf("%s",name);
}
```

**Output:**
Lachi (It only accepts the data upto the spaces)

**32. What is the difference between while(a) and while(!a)?**

while(a) means while(a!=0)

while(!a) means while(a==0)

**33. Why we don't use the symbol '&' symbol, while reading a String through scanf()?**

The '&' is not used in scanf() while reading string, because the character variable itself specifies as a base address.

**Example:** name, &name[0] both the declarations are same.

**34. What is the difference between static and auto storage classes?**

| | Static | Auto |
|---|---|---|
| **Storage** | Memory | Memory |
| **Initial value** | Zero | Garbage value |
| **Scope** | Local to the block in which the variables is defined | Local to the block in which the variable is defined. |
| **Life** | Value of the variable persists between different function calls. | The block in which the variable is defined. |

**35. What is the output of the program?**

```
main()
{
increment();
increment();
increment();
}
```

```
increment()
{
        static int i=1;
        printf("%d\n",i)
        i=i+1;
}
```

**OUTPUT:**

1 2 3

**36. Why header files are included in 'C' programming?**

- This section is used to include the function definitions used in the program.
- Each header file has 'h' extension and include using '# include' directive at the beginning of a program.

**37. List out some of the rules used for 'C' programming.**

- All statements should be written in lower case letters. Upper case letters are only for symbolic constants.
- Blank spaces may be inserted between the words. This improves the readability of statements.
- It is a free-form language; we can write statements anywhere between '{' and '}'.

```
a = b + c;
d = b*c;
        (or)
a = b+c; d = b*c;
```

- Opening and closing braces should be balanced.

**38. Define delimiters in 'C'.**

| | Delimiters | Use |
|---|---|---|
| **:** | Colon | Useful for label |
| **;** | Semicolon | Terminates Statement |
| **( )** | Parenthesis | Used in expression and functions |
| **[ ]** | Square Bracket | Used for array declaration |
| **{ }** | Curly Brace | Scope of statement |
| **#** | Hash | Preprocessor directive |
| **,** | Comma | Variable Separator |

**39. What do you mean by variables in 'C'?**

- A variable is a data name used for storing a data value.
- Can be assigned different values at different times during program execution.
- Can be chosen by programmer in a meaningful way so as to reflect its function in the program.
- Some examples are:

  Sum
  percent_1
  class_total

**40. List the difference between float and double datatype.**

| S No | Float | Double Float / Double |
|---|---|---|
| **1** | Occupies 4 bytes in memory | Occupies 8 bytes in memory |
| **2** | Range : 3.4 e-38 to 3.8e+38 | Range : 1.7 e-308 to 1.7e+308 |
| **3** | Format Specifier: % f | Format Specifier: % lf |
| **4** | **Example :** float a; | **Example :** double y;<br>There exists long double having a range of 3.4 e -4932 to 3.4 e +4932 and occupies 10 bytes in memory.<br>**Example:** long double k; |

**41. Differentiate break and continue statement**

| S No | break | continue |
|---|---|---|
| **1** | Exits from current block / loop | Loop takes next iteration |
| **2** | Control passes to next statement | Control passes to beginning of loop |
| **3** | Terminates the program | Never terminates the program |

**42. List the types of operators.**

| S No | Operators Types | Symbolic Representation |
|---|---|---|
| **1** | Arithmetic operators | = , - , * , / and % |
| **2** | Relational operators | > , < , == , >=, <= and != |
| **3** | Logical operators | && , \|\| and ! |
| **4** | Increment and Decrement operators | ++ and – |
| **5** | Assignment operators | = , + = , - = , * = , / = , ^ = , ; = , & = |
| **6** | Bitwise operators | & , \| , ^ , >> , << , and ~ |
| **7** | Comma operator | , |
| **8** | Conditional operator | ? : |

**43. Distinguish between while..do and do..while statement in C. (JAN 2009)**

| While..DO | DO..while |
|---|---|

| | |
|---|---|
| (i) Executes the statements within the while block if only the condition is true. | (i) Executes the statements within the while block at least once. |
| (ii) The condition is checked at the starting of the loop | (ii) The condition is checked at the end of the loop |

**44. Compare switch( ) and nestedif statement.**

| S No | switch( ) case | nested if |
|---|---|---|
| **1** | Test for equality ie., only constant values are applicable. | It can equate relational (or) logical expressions. |
| **2** | No two case statements in same switch. | Same conditions may be repeated for a number of times. |
| **3** | Character constants are automatically | Character constants are automatically converted to integers. |
| **4** | converted to integers. In switch( ) case statement nested if can be used. | In nested if statement switch case can be used. |

**45. Distinguish Increment and Decrement operators.**

| S No | Increment ++ | Decrement -- |
|---|---|---|
| **1** | Adds one to its operand | Subtracts one from its operand |
| **2** | Equivalent x = x + 1 | Equivalent x = x - 1 |
| **3** | Either follow or precede operand | Either follow or precede operand |
| **4** | **Example** : ++x; x++; | **Example** : --x; x--; |

**46. Give the syntax for the 'for' loop statement**

```
for (Initialize counter; Test condition; Increment / Decrement)
{
 statements;
}
```

- Initialization counter sets the loop to an initial value. This statement is executed only once.
- The test condition is a relational expression that determines the number of iterations desired or it determines when to exit from the loop. The 'for' loop continues to execute as long as conditional test is satisfied. When condition becomes false, the control of program exists the body of the 'for' loop and executes next statement after the body of the loop.

- The increment / decrement parameter decides how to make changes in the loop.
- The body of the loop may contain either a single statement or multiple statements.

**47. What is the use of sizeof( ) operator?**

- The sizeof ( ) operator gives the bytes occupied by a variable.
- No of bytes occupied varies from variable to variable depending upon its data types.

**Example:**

```
int x,y;
printf("%d",sizeof(x));
```

**Output:**

2

**48. What is a loop control statement?**

Many tasks done with the help of a computer are repetitive in nature. Such tasks can be done with loop control statements.

**49. What are global variable in 'C'?**

- This section declares some variables that are used in more than one function. such variable are called as global variables.
- It should be declared outside all functions.

**50. Write a program to swap the values of two variables (without temporary variable).**

```
#include <stdio.h>
#include <conio.h>
void main( )
{
        int a =5; b = 10;
        clrscr( );
        prinf("Before swapping a = %d b = %d ", a , b);
        a = a + b;
    B = a – b;
        a = a – b;
        prinf("After swapping a = %d b = %d", a,b);
        getch( );
}
```

**Output:**

Before swapping a = 5 b = 10
After swapping a = 10 b = 5

**51. Write short notes about main ( ) function in 'C' program. (MAY 2009)**

- Every C program must have main ( ) function.
- All functions in C, has to end with '( )' parenthesis.
- It is a starting point of all 'C' programs.
- The program execution starts from the opening brace '{' and ends with closing brace '}', within which executable part of the program exists.

# 12 MARKS

**1. Explain in detail about 'C' declarations and variables.**

In C, lowercase and uppercase characters are very important. All commands in C must be lowercase. The C programs starting point is identified by the word main( ). This informs the computer as to where the program actually starts.

The brackets that follow the keyword main indicate that there are no arguments supplied to this program.

The two braces, { and }, signify the begin and end segments of the program. The purpose of the statement

include <stdio.h> is to allow the use of the printf statement to provide program output. Text to be displayed by printf() must be enclosed in double quotes. The program has only one statement printf("Programming in C is easy.\n");

printf() is actually a function (procedure) in C that is used for printing variables and text. Where text appears in double quotes "", it is printed without modification. There are some exceptions however. This has to do with the \ and % characters. These characters are modifier's, and for the present the \ followed by the n character represents a newline character. Thus the program prints

**Programming in C is easy.**

and the cursor is set to the beginning of the next line. As we shall see later on, what follows the \ character will determine what is printed, ie, a tab, clear screen, clear line etc. Another important thing to remember is that all C statements are terminated by a semi-colon ;

**General rules of 'C' language:**

- program execution begins at main()
- keywords are written in lower-case
- statements are terminated with a semi-colon
- text strings are enclosed in double quotes
- C is case sensitive, use lower-case and try not to capitalize variable names
- \n means position the cursor on the beginning of the next line
- printf() can be used to display text to the screen
- The curly braces {} define the beginning and end of a program block.

## BASIC STRUCTURE OF C PROGRAMS

C programs are essentially constructed in the following manner, as a number of well defined sections.

```
/* HEADER SECTION */
/* Contains name, author, revision number*/
/* INCLUDE SECTION */
/* contains #include statements */
/* CONSTANTS AND TYPES SECTION */
/* contains types and #defines */
/* GLOBAL VARIABLES SECTION */
/* any global variables declared here */
/* FUNCTIONS SECTION */
/* user defined functions */
/* main() SECTION */
int main()
{
}
```

**A Simple Program**

The following program is written in the C programming language.

```
#include <stdio.h>
main()
{
printf("Programming in C is easy.\n");   }
```

**INITIALISING DATA VARIABLES AT DECLARATION TIME**

In C, variables may be initialized with a value when they are declared. Consider the following declaration, which declares an integer variable count which is initialized to 10. int count = 10;

**SIMPLE ASSIGNMENT OF VALUES TO VARIABLES**

The = operator is used to assign values to data variables. Consider the following statement, which assigns the value 32 an integer variable count, and the letter **A** to the character variable letter

```
count = 32;
letter = 'A'
```

**Variable Formatters**

%d decimal integer
%c character
%s string or character array
%f float
%e double

**HEADER FILES**

Header files contain definitions of functions and variables which can be incorporated into any C program by using the pre-processor #include statement. Standard header files are provided with each compiler, and cover a range of areas, string handling, mathematical, data conversion, printing and reading of variables.

To use any of the standard functions, the appropriate header file should be included. This is done at the beginning of the C source file. For example, to use the function printf() in a program, the line

#include <stdio.h> should be at the beginning of the source file, because the definition for printf() is found in the file stdio.h All header files have the extension .h and generally reside in the /include subdirectory.

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

The use of angle brackets <> informs the compiler to search the compilers include directory for the specified file. The use of the double quotes "" around the filename inform the compiler to search in the current directory for the specified file.

**2. Explain in detail about the constants, expressions and statements in 'C'.**

**1. Constants: (with examples)**

1. Numeric constants
   a. Integer Constants
   b. Real Constants
2. Character constants
   a. Single character Constants
   b. String Constants

**2. Expressions:**

An expression represents a single data item, such as number or a character. Logical conditions that are true or false are represented by expressions.

**Example:** a = p – q / 3 + r * 2 - 1

**3. Statements**

- Assignment Statements – Definition and examples
- Null Statements – Definition and examples
- Block of statements – Definition and examples
- Expression statements – Definition and examples
- Declaration statements – Definition and examples

**3. Discuss about the various data types in 'C'. (MAY 2009)**

The four basic data types are

**a. INTEGER**

These are whole numbers, both positive and negative. Unsigned integers (positive values only) are supported. In addition, there are short and long integers.

The keyword used to define integers is,

int

An example of an integer value is 32. An example of declaring an integer variable called **sum** is,

```
int sum;
sum = 20;
```

**b. FLOATING POINT**

These are numbers which contain fractional parts, both positive and negative. The keyword used to define float variables is,

float

An example of a float value is 34.12. An example of declaring a float variable called **money** is,

```
float money;
money = 0.12;
```

**c. DOUBLE**

These are exponentional numbers, both positive and negative. The keyword used to define double variables is,

double

An example of a double value is 3.0E2. An example of declaring a double variable called **big** is,

```
double big;
big = 312E+7;
```

**d. CHARACTER**

These are single characters. The keyword used to define character variables is,

char

An example of a character value is the letter **A**. An example of declaring a character variable called **letter** is,

```
char letter;
letter = 'A';
```

Note the assignment of the character A to the variable letter is done by enclosing the value in **single quotes**.

**Example:**

```
#include < stdio.h >
main()
{
int sum;
float money;
char letter;
double pi;
sum = 10; /* assign integer value */
money = 2.21; /* assign float value */
letter = 'A'; /* assign character value */
pi = 2.01E6; /* assign a double value */
printf("value of sum = %d\n", sum );
printf("value of money = %f\n", money );
printf("value of letter = %c\n", letter );
printf("value of pi = %e\n", pi );
}
```

**Sample program output**

```
value of sum = 10
value of money = 2.210000
value of letter = A
value of pi = 2.010000e+06
```

4. **Describe the various types of operators in 'C' language along with its priority.**

An ex pr e s s I o n is a sequence of operators and operands that specifies computation of a value, or that designates an object or a function, or that generates side effects, or that performs a combination thereof.

**1. ARITHMETIC OPERATORS:**

The symbols of the arithmetic operators are:-

| Operation | Operator | Comment | Value of Sum before | Value of sum after |
|---|---|---|---|---|
| Multiply | * | sum = sum * 2; | 4 | 8 |
| Divide | / | sum = sum / 2; | 4 | 2 |
| Addition | + | sum = sum + 2; | 4 | 6 |
| Subtraction | - | sum = sum -2; | 4 | 2 |
| Increment | ++ | ++sum; | 4 | 5 |
| Decrement | -- | --sum; | 4 | 3 |
| Modulus | % | sum = sum % 3; | 4 | 1 |

**Exam ple:**

```
#include <stdio.h>
main()
{
int sum = 50;
float modulus;
modulus = sum % 10;
printf("The %% of %d by 10 is %f\n", sum, modulus);
```

}

**PRE/POST INCREMENT/DECREMENT OPERATORS**

PRE means do the operation first followed by any assignment operation. POST means do the operation after any assignment operation. Consider the following statements ++count; /* PRE Increment, means add one to count */ count++; /* P OST Increment, means add one to count */

**Example:**

```
#include <stdio.h>
main()
{
int count = 0, loop;
loop = ++count; /* same as count = count + 1; loop = count; */
printf("loop = %d, count = %d\n", loop, count);
loop = count++; /* same as loop = count; count = count + 1; */
printf("loop = %d, count = %d\n", loop, count);
}
```

If the operator precedes (is on the left hand side) of the variable, the operation is performed first, so the statement

loop = ++count;

really means increment count first, then assign the new value of count to loop.

**2. <u>THE RELATIONAL OPERATORS</u>**

These allow the comparison of two or more variables.

= = equal to
! = not equal
< less than
< = less than or equal to
> greater than
> = greater than or equal to

**Example:**

```
#include <stdio.h>
main() /* Program introduces the for statement, counts to ten */
{
int count;
for( count = 1; count <= 10; count = count + 1 )
printf("%d ", count );
printf("\n");
}
```

**<u>3. LOGICAL OPERATORS (AND, NOT, OR, EOR)</u>**

**Combining more than one condition**

These allow the testing of more than one condition as part of selection statements. The symbols are

**LOGICAL AND &&**

Logical and requires all conditions to evaluate as TRUE (non-zero).

**LOGICAL OR ||**

Logical or will be executed if any ONE of the conditions is TRUE (non-zero).

**LOGICAL NOT !**

logical not negates (changes from TRUE to FALSE, vsvs) a condition.

**LOGICAL EOR ^**

Logical eor will be excuted if either condition is TRUE, but NOT if they are all true.

**Example:**

The following program uses an if statement with logical AND to validate the users input to be in the range 1-10.

```
#include <stdio.h>
main()
{
int number;
int valid = 0;
while( valid == 0 ) {
printf("Enter a number between 1 and 10 →");
scanf("%d", &number);
if( (number < 1 ) || (number > 10) ){
printf("Number is outside range 1-10. Please re-enter\n");
valid = 0;
}
else
valid = 1;
}
printf("The number is %d\n", number );
}
```

**Example:**

**NEGATION**

```
#include <stdio.h>
main()
{
int flag = 0;
if( ! flag ) {
printf("The flag is not set.\n");
flag = ! flag;
}
printf("The value of flag is %d\n", flag);
}
```

**Example:**

Consider where a value is to be inputted from the user, and checked for validity to be within a certain range, lets say between the integer values 1 and 100.

```
#include <stdio.h>
main()
{
int number;
int valid = 0;
while( valid == 0 ) {
printf("Enter a number between 1 and 100");
scanf("%d", &number );
if( (number < 1) || (number > 100) )
printf("Number is outside legal range\n");
else
valid = 1;
}
printf("Number is %d\n", number );
```

}

**4. THE CONDITIONAL EXPRESSION OPERATOR or TERNARY OPERATOR**

This conditional expression operator takes THREE operators. The two symbols used to denote this operator are the ? and the :. The first operand is placed before the ?, the second operand between the ? and the :, and the third after the :. The general format is,

**condition ? expression1 : expression2.**

If the result of condition is TRUE ( non-zero ), expression1 is evaluated and the result of the evaluation becomes the result of the operation. If the condition is FALSE (zero), then expression2 is evaluated and its result becomes the result of the operation. An example will help,

```
s = ( x < 0 ) ? -1 : x * x;
```

If x is less than zero then s = -1
If x is greater than zero then s = x * x

**Example:**

```
#include <stdio.h>
main()
{
int input;
printf("I will tell you if the number is positive, negative or zero!"\n");
printf("please enter your number now-→");
scanf("%d", &input );
(input < 0) ? printf("negative\n") : ((input > 0) ? printf("positive\n") :
printf("zero\n"));
}
```

**5. BIT OPERATIONS**

| Operation | Operator | Comment | Value of Sum before | Value of sum after |
|---|---|---|---|---|
| AND | & | sum = sum & 2; | 4 | 0 |
| OR | \| | sum = sum \| 2; | 4 | 6 |
| Exclusive OR | ^ | sum = sum ^ 2; | 4 | 6 |
| 1's Complement | ~ | sum = ~sum; | 4 | -5 |
| Left Shift | << | sum = sum << 2; | 4 | 16 |
| Right Shift | >> | sum = sum >> 2; | 4 | 0 |

C has the advantage of direct bit manipulation and the operations available are,

**Example:**

```
/* Example program illustrating << and >> */
#include <stdio.h>
main()
{
int n1 = 10, n2 = 20, I = 0;
I = n2 << 4; /* n2 shifted left four times */
printf("%d\n", i);
I = n1 >> 5; /* n1 shifted right five times */
printf("%d\n", i);
}
```

**Example:**

```
/* Example program using EOR operator */
#include <stdio.h>
main()
{
int value1 = 2, value2 = 4;
value1 ^= value2;
value2 ^= value1;
value1 ^= value2;
printf("Value1 = %d, Value2 = %d\n", value1, value2);
}
```

**Example:**

```
/* Example program using AND operator */
#include <stdio.h>
main()
{
int loop;
for( loop = 'A'; loop <= 'Z'; loop++ )
printf("Loop = %c, AND 0xdf = %c\n", loop, loop & 0xdf);
}
```

**5. Explain about the various decision making statements in 'C' language.**

**(JAN 2009/FEB2010)**

## DECISION MAKING

### 1. IF STATEMENTS

The if statements allows branching (decision making) depending upon the value or state of variables. This allows statements to be executed or skipped, depending upon decisions.

The basic format is,

```
if( expression )
program statement;
```

**Example:**

```
if( students < 65 )
++student_count;
```

In the above example, the variable student_count is incremented by one only if the value of the integer variable students is less than 65. The following program uses an if statement to validate the users input to be in the range 1-10.

**Example:**

```
#include <stdio.h>
main()
{
int number;
int valid = 0;
while( valid == 0 ) {
printf("Enter a number between 1 and 10 →");
scanf("%d", &number);
/* assume number is valid */
valid = 1;
if( number < 1 ) {
printf("Number is below 1. Please re-enter\n");
valid = 0;
}
if( number > 10 ) {
printf("Number is above 10. Please re-enter\n");
valid = 0;
}
}
printf("The number is %d\n", number );
}
```

**2. IF ELSE**

The general format for these are,

```
if( condition 1 )
statement1;
else if( condition 2 )
statement2;
else if( condition 3 )
statement3;
else
statement4;
```

The else clause allows action to be taken where the condition evaluates as false (zero). The following program uses an if else statement to validate the users input to be in the range 1-10.

**Example:**

```
#include <stdio.h>
main()
{
int number;
int valid = 0;
while( valid == 0 ) {
printf("Enter a number between 1 and 10 →");
scanf("%d", &number);
if( number < 1 ) {
printf("Number is below 1. Please re-enter\n");
valid = 0;
}
else if( number > 10 ) {
printf("Number is above 10. Please re-enter\n");
valid = 0;
```

```
}
else
valid = 1;
}
printf("The number is %d\n", number );
}
```

This program is slightly different from the previous example in that an else clause is used to set the variable valid to 1. In this program, the logic should be easier to follow.

**3. NESTED IF ELSE**

/* Illustates nested if else and multiple arguments to the scanf function. */

**Example:**

```
#include <stdio.h>
main()
{
int invalid_operator = 0;
char operator;
float number1, number2, result;
printf("Enter two numbers and an operator in the format\n");
printf(" number1 operator number2\n");
scanf("%f %c %f", &number1, &operator, &number2);
if(operator == '*')
result = number1 * number2;
else if(operator == '/')
result = number1 / number2;
else if(operator == '+')
result = number1 + number2;
else if(operator == '-')
result = number1 – number2;
else
invalid_operator = 1;
if( invalid_operator != 1 )
printf("%f %c %f is %f\n", number1, operator, number2, result );
else
printf("Invalid operator.\n");
```

**6. Write short notes on the following: (JAN 2009)**

'for' loop
'while' loop
'dowhile' loop
'Switch case ' **(MAY 2009/FEB 2009/FEB 2010)**

## BRANCHING AND LOOPING

**1. ITERATION, FOR LOOPS**

The basic format of the for statement is,

```
for( start condition; continue condition; re-evaulation )
program statement;
```

**Example:**

```
/* sample program using a for statement */
#include <stdio.h>
main() /* Program introduces the for statement, counts to ten */
{
int count;
for( count = 1; count <= 10; count = count + 1 )
printf("%d ", count );
printf("\n");
}
```

The program declares an integer variable count. The first part of the for statement **for (count = 1; initialized the value of count to 1.**

The for loop continues with the condition count <= 10; evaluates as TRUE. As the variable count has just been initialized to 1, this condition is TRUE and so the program statement printf("%d ", count ); is executed, which prints the value of count to the screen, followed by a space character.

Next, the remaining statement of the for is executed count = count + 1); which adds one to the current value of count. Control now passes back to the conditional test, count <= 10; which evaluates as true, so the program statement printf("%d ", count ); is executed.

Count is incremented again, the condition re-evaluated etc, until count reaches a value of 11.

When this occurs, the conditional test count <= 10; evaluates as FALSE, and the for loop terminates, and program control passes to the statement printf("\n"); which prints a newline, and then the program terminates, as there are no more statements left to execute.

## 2. THE WHILE STATEMENT

The while provides a mechanism for repeating C statements whilst a condition is true. Its format is, while( condition ) program statement;

Somewhere within the body of the while loop a statement must alter the value of the condition to allow the loop to finish.

**Example:**

```
/* Sample program including while */
#include <stdio.h>
main()
{
int loop = 0;
while( loop <= 10 ) {
printf("%d\n", loop);
++loop;
}
}
```

The above program uses a while loop to repeat the statements

**printf("%d\n",loop); ++loop;** the value of the variable loop is less than or equal to 10.

## 3. THE DO WHILE STATEMENT

The do { } while statement allows a loop to continue whilst a condition evaluates as TRUE (non-zero). The loop is executed as least once.

**Example:**

```
/* Demonstration of DO...WHILE */
#include <stdio.h>
main()
{
int value, r_digit;
printf("Enter the number to be reversed.\n");
scanf("%d", &value);
do {
r_digit = value % 10;
printf("%d", r_digit);
value = value / 10;
        } while( value != 0 );
printf("\n");
}
```

The above program reverses a number that is entered by the user. It does this by using the modulus % operator to extract the right most digit into the variable r_digit. The original number is then divided by 10, and the operation repeated whilst the number is not equal to 0.

4. **SWITCH CASE:**

The switch case statement is a better way of writing a program when a series of if elses occurs.

The general format for this is,

```
switch ( expression ) {
case value1:
program statement;
program statement;
.....
break;
case valuen:
program statement;
.......
break;
default:
.......
.......
break;
}
```

The keyword break must be included at the end of each case statement. The default clause is optional, and is executed if the cases are not met. The right brace at the end signifies the end of the case selections.

**Example:**

```
#include <stdio.h>
main()
{
int menu, numb1, numb2, total;
```

```
printf("enter in two numbers →");
scanf("%d %d", &numb1, &numb2 );
printf("enter in choice\n");
printf("1=addition\n");
printf("2=subtraction\n");
scanf("%d", &menu );
switch( menu ) {
case 1: total = numb1 + numb2; break;
case 2: total = numb1 – numb2; break;
default: printf("Invalid option selected\n");
}
if( menu == 1 )
printf("%d plus %d is %d\n", numb1, numb2, total );
else if( menu == 2 )
printf("%d minus %d is %d\n", numb1, numb2, total );
}
```

The above program uses a switch statement to validate and select upon the users input choice, simulating a simple menu of choices.

**7. Explain briefly about the input and output function in 'C'. (MAY 2009/FEB 2009)**

## MANAGING INPUT AND OUTPUT OPERATORS

**1 printf ():**

printf() is actually a function (procedure) in C that is used for printing variables and text. Where text appears in double quotes "", it is printed without modification. There are some exceptions however.

This has to do with the \ and % characters. These characters are modifiers, and for the present the \ followed by the n character represents a newline character.

**Example:**

```
#include <stdio.h>
main()
{
printf("Programming in C is easy.\n");
printf("And so is Pascal.\n");
}
@ Programming in C is easy.
And so is Pascal.
```

**FORMATTERS for printf are,**

**Cursor Control Formatters**

```
\n newline
\t tab
\r carriage return
\f form feed
\v vertical tab
```

**2. Scanf ():**

Scanf () is a function in C which allows the programmer to accept input from a keyboard.

**Example:**

```
#include <stdio.h>
main() /* program which introduces keyboard input */
{
int number;
printf("Type in a number \n");
scanf("%d", &number);
printf("The number you typed was %d\n", number);
}
```

**FORMATTERS FOR scanf()**

The following characters, after the % character, in a scanf argument, have the following effect.

D read a decimal integer
o read an octal value
x read a hexadecimal value
h read a short integer
l read a long integer
f read a float value
e read a double value
c read a single character
s read a sequence of characters
[...] Read a character string. The characters inside the brackets

**3. ACCEPTING SINGLE CHARACTERS FROM THE KEYBOARD**

**Getchar, Putchar**

getchar() gets a single character from the keyboard, and putchar() writes a single character from the keyboard.

**Example:**

The following program illustrates this,

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

{
int i;
int ch;
for( i = 1; i<= 5; ++i ) {
ch = getchar();
putchar(ch);
}
}
```

The program reads five characters (one for each iteration of the for loop) from the keyboard. Note that getchar() gets a single character from the keyboard, and putchar() writes a single character (in this case, ch) to the console screen.

**8. (a) Describe in detail about type conversions in 'C' with example.**

**(b) Define delimiters. List them. Give an example program using various delimiters.**

9. **Explain the following:**
   - Keywords
   - Identifiers
   - C character set
   - Constant and Volatile variables.

10. **Explain the following:**
   - break statement with example program
   - continue statement with example program
   - goto statement with example program

**********