Data structure: A data structure is a specialized format for organizing, processing, retrieving and storing data.

Types of data structures: The data structure type used in a particular situation is determined by the type of operations that will be required for the kind of algorithms that will be applied. The various data structure types include the following:

Array: An array stores a collection of elements in a same data type. Items that are the same type are stored together so the position of each element can be calculated on retrieved easily by an index. Array can be fixed on flexible in length.

Stack: A stack stores a collection of items in the linear order that operations are applied. This order could be last in, first out (LIFO) on first in first out (FIFO)

Queue: A queue stores a collection of items like a stack. however, the operation order can only be first in first out.

Linked list: A linked list stores a collection of items in a linear order. Each element on node in a linked list contains a data items. as well as a reference, on link, to the next item in the list.

Tree: A tree stores a collection of items in an abstract, hierarchical way. Each node is associated with a key value,

with parent nodes linked to child ~~nodes~~ or subnodes nodes.
There is one root node that is the ancestor of all the nodes in the tree.

Heap: A heap is a tree based structure in which each Parent nodes associated key value is greater than or equal to the key values of any of its childrens key values.

Graph: A graph is stores a collection of items in a non-linear fashion. Graphs are made up of a finite set of nodes, also known as vertices, and lines that connect them, also known as edges. These are useful for representing real-world systems such as computer networks.

Trie: A trie, also known as a ~~keyboard~~ keyword tree, is a data structure that stores strings as data ~~to~~ items that can be organised in a visual graph.

Hash table: A hash table, also known as hash map, stores a collection of items in an associative array that plots keys to values. A hash table uses a hash function to convert an index into an array of buckets that contain the desired data item.

Chapter 8 // Array ——

Data type – Primary or fundamental data type – see chapter 3)

Derived data type - Derived data types are those that are defined in term of other data types.

Array, pointers, are derived data types.

Derived data type can be four type.

(i) Function
(ii) Array
(iii) Pointers
(iv) References

User-defined data type :- The data types which are defined by users is known as user defined data type. These type include.

(i) Class
(ii) structure
(iii) Union
(iv) Enumeration
(v) Typedef defined data type.

Array: An array is a data structure that contains a group of elements of the same data type. Arrays are commonly used in computer programs to organize data so that a related set of values can be easily sorted or searched.

Dimension

Various types of arrays are.

One dimentional array: In one dimentional array the elements are stored in ~~contigu~~ Adjacent (सजिएका) memory locations where each element is accessed by using a single index value. It is linear data structure storing all the elements in sequence.

example: int number[5];
int mark[] = {85, 90, 60, 93, 98};

here number and mark are the one-D array.

Two dimentional Array: The two dimentional array is used for representing the elements of the array in the form of rows and columns and these are used for representing the matrix.

Example int a[3][4];
int table[][3] = {{1, 2, 3},
{4, 5, 6}};

Here a and table is two dimentional array.

Multi dimentional array: C allows arrays of three or more dimentioned. The exact limit is determined by the compiler.

Dimension array

The general multi dimentional array is

type array_name[s1][s2]....[sm];

where si is the size of ith dimention.

int survey[3][5][12];

survey is three dimension array.

Character array of string: Like an integer characters are also be in the array. The array of characters are called as the string. They are generally used for representing the string. String is always terminated with the null character.

Compile time initialization of 1D

(Definition of array) We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.

The general form of initialization of array is:

type array_name[size] = { list of values };

The values in the list are separated by commas. For example statement,

int number[5] = { 1, 10, 9, 2, 11 };

We declare number array and assign all the values. If the number of values are less than number of elements, then remaining elements are intitialized by '0'.

Some examples,

```
int age[] = {20, 18, 19, 50};
int marks[5] = {32, 33};
```

If the number of values is greater than number of elements then it will not work.

Example:

```
int number[3] = {10, 9, 8, 5, 2};
```

is illigal in C

Run time initialization: Array can be initialized at run time. We use scanf() to initialize an array in the run time.

The general form for intiallization is

Example:

For small array - int a[3];

```
scanf("%d %d %d", &a[0], &a[1], &a[2]);
```

if the size of array is large, int number[50];

```
for(int i=0; i<50; i++)
    scanf("%d", &number[i]);
```

Here we consider the size is 50.

for V example

The general form is;

type array_name[size];

```
for(int i=0; i<size; i++)
    scanf("format_specifier_of_that_type",
          &Index_of_array);
```

We can enter the value of array from keyboar

2 marks answer: for more marks add the definition

Initializing 2D array: Like 1D array two dimensional array may be initialized by following their declaration with a list of values, enclosed in braces.

For example int table[2][3] = {0,0,0,1,1,1};

Here, we initialized the first row to zero and to second row to one.

We can also initialized the element in the form of braces. For example;

```
Table[2][3] = {
                {0,0,0},
                {2,2,2}
              };
```

We may also pentially initialized a 2-D array then the remaining element are automatically set to zero.

```
int table[2][3] = {
                  {1, 1},
                  {2}
                  };
```

Mean by dynamic arrays: When we postpone the ~~memory~~ array definition and allocate memory for the array at run time are is known as as dynamic array

An array created at compile time by specifying size in the source code has a fixed size and can't be modified at run time. The process of allocating memory at compile time is known as static memory memory allocation and such array is ~~static~~ static array. But in C it is possible to allocate memory to arrays at run time. This feature is known as dynamic memory allocation and the arrays created at run time are called dynamic arrays.

This process effectively postpones the array definition to run time.

Dynamic arrays are created using pointers. Pointer variables and memory management function (malloc, calloc, realloc) are used for creating dynamic array.

Chapter Character Arrays and Strings

String handling function:

- strcat() - concatenates two strings
- strcmp() - Compare two strings
- strcpy() - copies one string over another
- strlen() - finds the length of a string

What is string: In C programming, a string is sequence of characters terminated with a null character (\0).

For example: char c[10] = "c string";

Index → 0 1 2 3 4 5 6 7 8 9

c[10]	c	' '	s	t	r	i	n	g	\0

What are the common operations performed on character string:

In C programming, a string is a sequence of characters terminated with a null character (\0). String are defined as an array of characters. Some of the most commonly used string operations are as follows:

strcat(): The strcat() function will append a copy of the source string to the end of destination string. The strcat() function takes two arguments and they are dest and src. It will append copy of the source string in the destination string.

Ex: ~~Syntax:~~ strcat(dest, src);

strrchr(): strrchr is a predefined function used for string handling. (string is the header file required for string functions. This functions returns a pointer to the last occurance of a character in a string.

~~Syntax~~ Syntax: strrchr(const char *str, int c);

Example: char *ptr = strrchr(string, character);

strcmp(): strcmp() is a build in library function and it is declared in <string.h> header file. This function takes two strings as arguments and compare these two strings.

Ex. Syntax: int value = strcmp(str1, str2);

This function takes two strings as parameter and returns an integer value based on the comparison of string.

strcpy(): strcpy() is a standard library function is use to copy one string to another. In C it is present in string.h header file.

Example: strcpy(str2, str1)

Here, elements of str1 is coppied to string str2.

strlen(): The strlen() function calculates the length of a given string. The strlen() function is defined in string.h header file. It doesn't count null character.

Example: ~~strlen(str);~~

int length = strlen(str);

strlen() return the length of the string str.

Why do we need a terminating null character

The null character in the c programming language is used to terminate the character string. In other words, the null character is used to represent the end of the string. The end of the character string or the null byte is represent by '\0' or simply NULL.

The NULL character is used for determined the length of a string. It also means that a string can not contains a null there is a NULL in memory but it is after the last character, not in the string.

Does C support String data type?

→ The C Language does not provide an inbuild data type for strings but it has an access specifier "%s" which can be used to directly print and read strings.

example:

```
#include <stdio.h>
int main()
{
    char str[50];   declaring string
    scanf("%s", str);   // reading string
    printf("%s", str);   // print string
    return 0;
}
```

Chapter - 10 / User-Defined Functions

Category of functions : Generally a function contain function name, function arguments, return type and also definition of this function. Based on these categories ~~functions~~ function may belong on of the following categories.

Function with no arguments and no return values: When a function has no arguments. It does not receive any data from the calling function. Similarly, when it does not return a value, the calling function doesn't receive any data from the called function.

Example,

```
void function () {
    ------
}
main ()
{
    ------
    function ();
    ------
}
```

Function with Arguments but no return value: When a function has no return value. But there present a arguments, or function parameters.

Example:

```
void function(a1, a2, a3 ... an);
{
  ...
}
main()
{
  ...
  function(f1, f2, f3 ... fn);
  ...
}
```

Formal arguments.

actual arguments

a array of character or integer

Function with arguments and a return value:

In this type of functions are receives data from calling function and called function return a value to the calling function.

Example:

```
return_type function(f.){
  ...
  return(e);
}
main()
{
  ...
  variable = function(a);
  ...
}
```

*Here, return_type means a data_type, which data type a function needs to return.

Function with no arguments but a returns a value

This type of function is important where we need to design a function, that takes no value from the calling function but return a value to calling function.

Example:

```
return_type function() {
    ...
    return(e);
}
main()
{
    ---
    variable = function();
    ---
}
```

Here, return types means a data type, which data type need to return the called function.

Function with that return multiple values,

When a function have to return multiple values, that time we design this function with pointer to variable. Using pointer we can be return multiple values.

9. What do you mean by searching and sorting in programming.

→ searching: searching is an operation that is used when a look-up needs to be conducted on a set of data to locate a particular element. There are many ways to search a number. linear search and binary search is most popular. Binary search is very efficient way, it doesn't look every element. The time complexity of binary search is Order of Logn. O(Logn). But it is only works when the array or set of data is stored on sorted order.

When we have to a search an unsorted element in set of data, then linear search is very helpful. It looks works by looking for the requires element at each position of the array, starting with the first and moving on in a sequence. The time complexity of search a search process is O(n). Example:

```
#include<stdio>
int linear_search (int a[], int n, int Value) {
    int i;
    for(i=0; i<n; i++)
        if(a[i] == value)
            return i; //search successfull
    return -1; //search unsuccessfull
}
int main()
{ int array[5] = {2, 10, 11, 9, 6};
  if(linear_search(array, 5,
```

```
int size = 5;
int data = 10; // find the index;
int n = linear_search(array, size, data);
if(linear_search(array, size, data) == -1)
    printf("Data not found in the array");
else
    printf("The data locate at index : %d",
}                                linear_search(array, size, data));
```

Sorting: Sorting is an operation that is needed to arrange the data set in a specified order. Bubble sort, merge sort, heap sort, radix sort, quick sort selection sort are a few sorting algorithms. We now consider a well-known sorting algorithm named selection sort - to arrange a set of numbers in ascending order. The algorithm chooses minimum among all elements and places it at the first position. Now this chooses the second minimum number and place in the second position and so on.

```
#include<stdio.h>
void selection_sort(int array[], int n)
{   int i, index;
    for(i=0; i<n; i++)
    {  index = i;
       for(j=i+1; j<n; j++)
          if(array[index] > array[j])
             index = j;
```

```
        if(index != i){
            int temp = arra[i];
            arra[i] = arra[index];
            arra[index] = temp;
        }
    }
}
int main()
{
    int arra[5] = {9, 10, 2, 8, 1};
    int size = 5;
    selection_sort(arra, size);
    printf("After sorting the array is: ");
    for(int i=0; i<size; i++)
        printf("%d ", arra[i]);
    return 0;
}
```

Here in the first loop
1 10 2 8 9
second loop
1 2 10 8 9
third loop
1 2 8 10 9
fourth loop
1 2 8 9 10

Q. Describe the scopes, visibility and lifetime of variables.

→ In c programming variables have a storage class according to storage class variables storage classes may be local variables, Global variables, static variables and Register variables. This classes define the scope, visibility and lifetime of variables.

The scope of variables determine over what region of the program a variable is actually available for use. If we try to use this variable in another region, there a error encountered the program.

Visibility refers to the accessibility of a variable from the memory.

Lifetime or longevity refers to the period during during which a variable retains a given value during execution of a program. so lifetime has a direct effect on the effectiveness (utility) of a given variable.

Q. What is prototype? Why it is necessary?

→ Prototype: A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.

Prototype is necessary when we declare a function for later use. A function prototype gives information to the compiler that the function may later be used in the program. We declare prototype in the global declaration section. Such declarations are available for all the

functions in the program. When we declared in a function definition, the prototype is called local prototype. ~~The~~ If a function have no

The moral is If a prototype is not available, ~~we will assume that the return type is an integer and that the type of the formal definition. This is wrong, the linker will fail and~~ we have to change the program. The moral is that we must always include prototype declarations, preferably in global declaration section.

9. Describe the two ways of passing parameters to functions. When do you prefer to use each of them?

→ There are two ways in which we can pass the parameters to the functions.

1. Call by value ① Here the values of the variables are passed by the calling function to the called function.

② If any value of the parameter in the called function has to be modified the change will be reflected only in the called function.

(iii) This happens as all the changes are made on the copy of the variables, and not on the actual ones.

Call by references: (i) Here, the address of the variables are passed by the calling function to the called function.

(ii) The address which we is used inside the function is used to access the actual argument used in the call.

(iii) If there are any changes made in the parameter they affect the passed argument.

(iv) For passing a value to the reference, the argument pointers are passed to the functions just like any other values.

When we directly want to change the calling function value by the called function, that time we use ~~pass~~ call by reference technique. This is also useful when a function needs to return multiple values.

pass by value is very good technique when ~~we have~~ a program need operations but values of ~~all~~ local variables are remain unchanged. ~~This~~ it doesn't directly effect the calling function values.

Difference between actual arguments and formal arguments.

Actual Arguments: ① Arguments which are mentioned in the function to call a function is known as actual arguments. On the other hand Arguments which are mentioned in function definition are formal arguments.

② Actual arguments are the values which called to the function where as formal arguments are used to just hold the value that is sent by calling function.

③ Examples:

```
function1(f1, f2, ... fn){
                formal arguments
}
main()
{
    function1(a1, a2, a ... an);
                actual arguments
}
```

Difference between "&" and "*" operator

→ The "&" is a unary operator in C ~~programming~~ which returns the memory address of the passed operand. This is also known as address of operator.

The * is a unary operator which returns the value of object pointed by a pointer variable. It is known as value of operator.

~~In a~~ In the time of function call. '&' used in the calling function to mentioned the address of the following variable. But '*' used in the called function to receive the address of calling function as ~~matching~~ pointer variable in formal argument.

Chapter-6 Decision Making and Branching:

Decision making

If.... Else Statements: These are decision making statements. It is used for checking certain condition to decide which block of code to be executed. The General syntax for if...else statement is as follows:

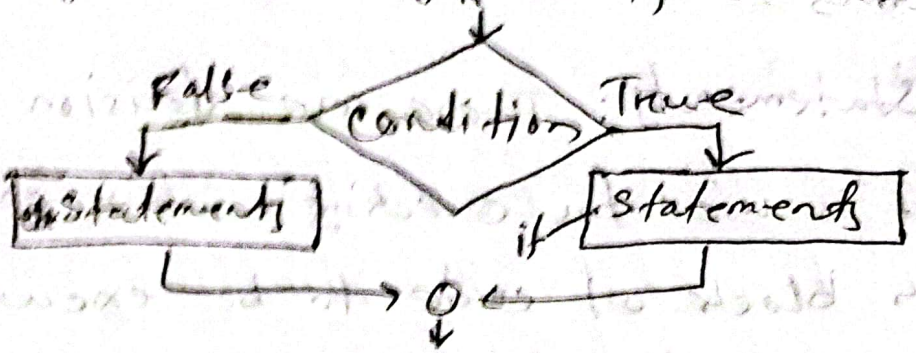
```
if (condition){
    expression or statements;
}
else{
    expression or statements;
}
```

When none of the condition is true we use else if at a time desirable number of times.

Example:

```
if (condition){
    statements;
}
else if (condition){
    statements;
}
else{
    statements;
}
```

The flowchart of the if condition is as follows.



If statement can be single if condition, without else part. This is use when there is no statement condition for false condition.

Example:
```
if(condition) {
    statements;
}
```

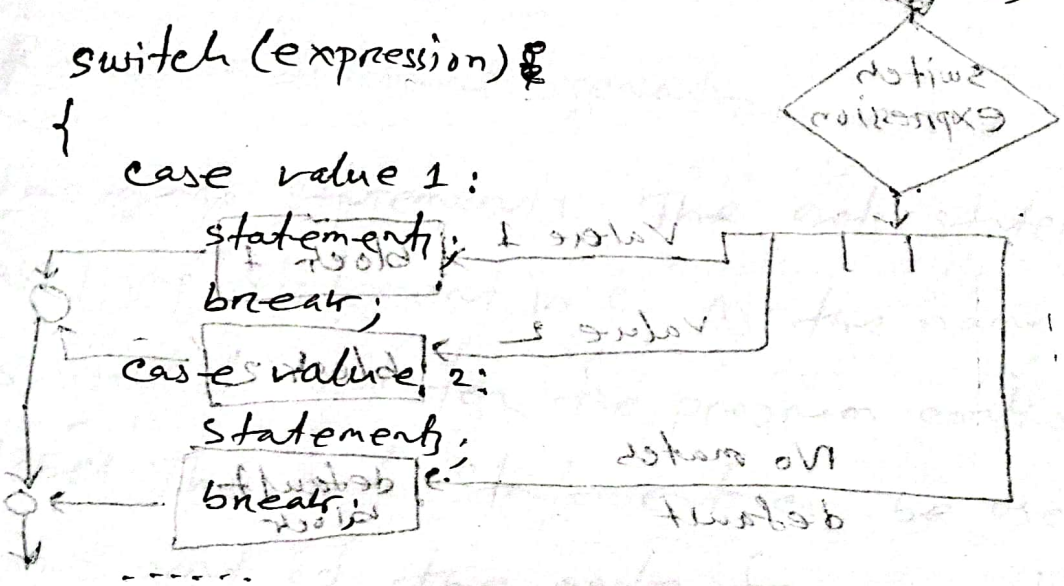
Nested if...else: When if statement contains another if...else statements is called nested if...else.

Example:
```
if(condition) {
    if(condition){
        statements;
    }
    else {
        statements;
    }
}else {
    statements;
}
```

Switch statement

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possibles values of a single variable called switch variable. The syntax of switch statements in C language is given bellow.

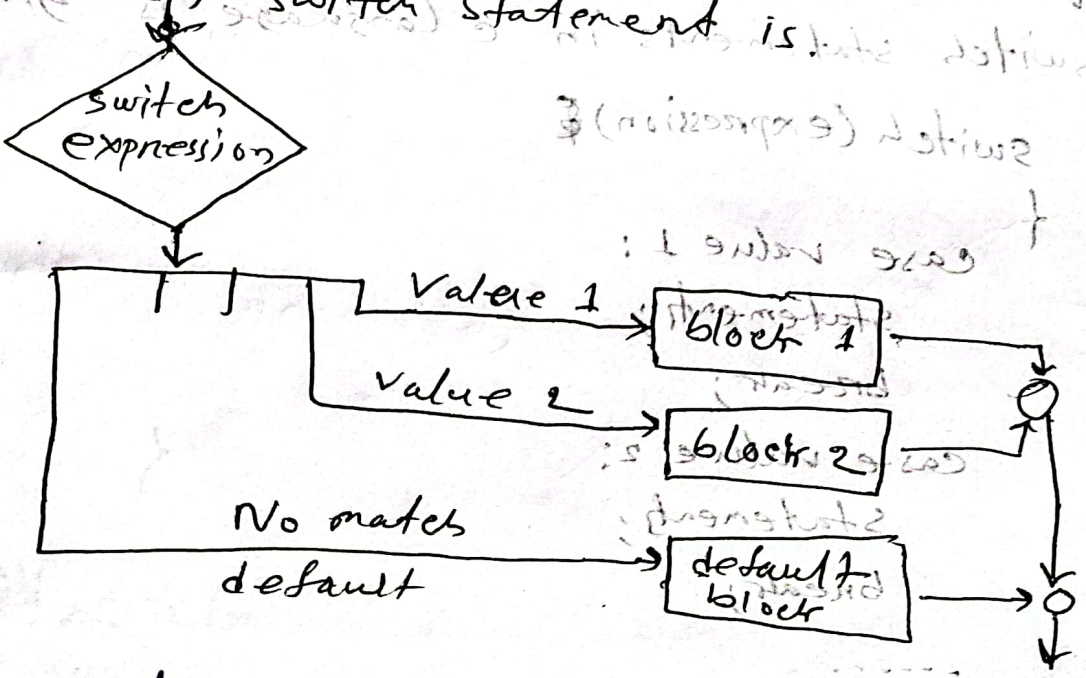
```
switch (expression)
{
   case value 1:
      statements;
      break;
   case value 2:
      statements;
      break;
   .....
   .....
   default :    // if all the cases are not matches
      statements;
}
```



The expression is an integer expression or characters. value 1, value 2 ... are constant or constant expressions. The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement.

The default is an optional case. When present, it will be executed if the value of the expression does not match with any of the case values.

Flow chart of switch statement is:



Rules for switch statements:

① The switch expression must be an integer or character type

② The case value must be constant or constant expression.

③ Case value must be unique.

④ Case value must end with colon.

⑤ The break statement transfer the control out of the switch statement.

~~but~~ (vi) Break statement is optional.

(vii) The default label is optional. ~~That is two or~~ if present, it will be executed when the expression does not find a matching case label/value.

(viii) There can be at most one default label.

(ix) It is permitted to nest switch statements.

Read conditional operator

The goto statement: The goto statement in known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statement can be used to repeat some part of the code for a particular condition.

syntax of goto:

```
label:
//some part of the code
goto label;
```

Example;

(in the next page)

```
main()
{
    int countt = 0;
repeated:
    Printf("Repeating loop %d\n", countt);
    countt++;
    if(countt==10)
        exit(0);
    goto repeated;
}
```

Chapter-6,7 Decision making and looping

The purpose of exit statement or exit() function
→ Exit is a jump statement in c programming language, which can takes an integer (zero or no zero) to represent different exit status.
Two types of exit status are
(i) Exit success: Exit success is indicated by exit(0) statement which means successful termination of the program. i.e. program has been executed without any error.
(ii) Exit Failure: Exit failure is indicated by exit(1) which means the abnormal termination of the program, i.e. some error occurred. We can use different integer other than 1 to indicate different types of error.

Example:

```
int main()
{
  file *fp;
  fp = fopen("myfile.txt", "r");
  if (fp == NULL) {
    printf("Error occured.");
    exit(1); // exit failure
  }
  exit(0); // exit success
```

0, 1 ছাড়া অন্য যেকোনো value দিতে হয়।

If we want to use exit statement, we must include the stdlib.h header.

General format of different loops:

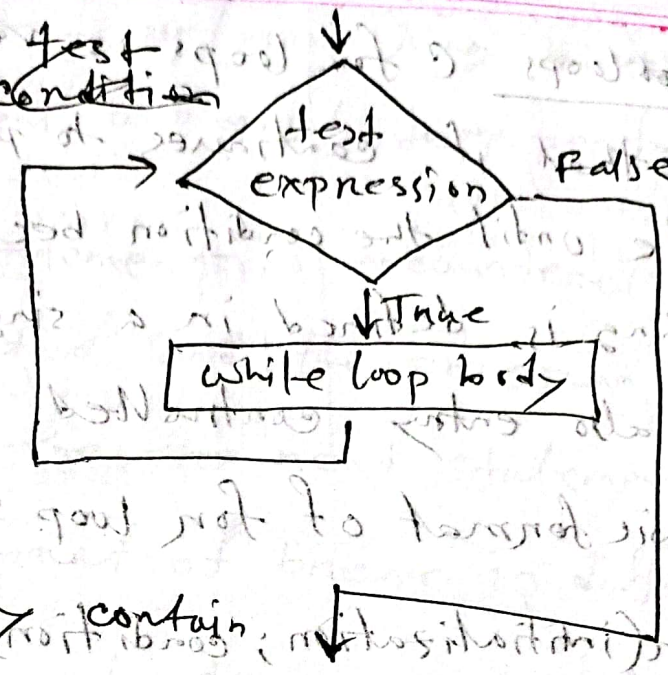
While: C While loops statement allows to repeatedly run the same blocks of code until a condition is met. While loop is a most basic loop in C programming. While loop has one control condition, and executes as long as the condition is true.

The basic format of while loop statement is

```
While (condition)
{
    statement;
    Increment or decrement;
}
```

```
While (test condition)
{
    body of the loop
}
```

Flow chart of the while loop is



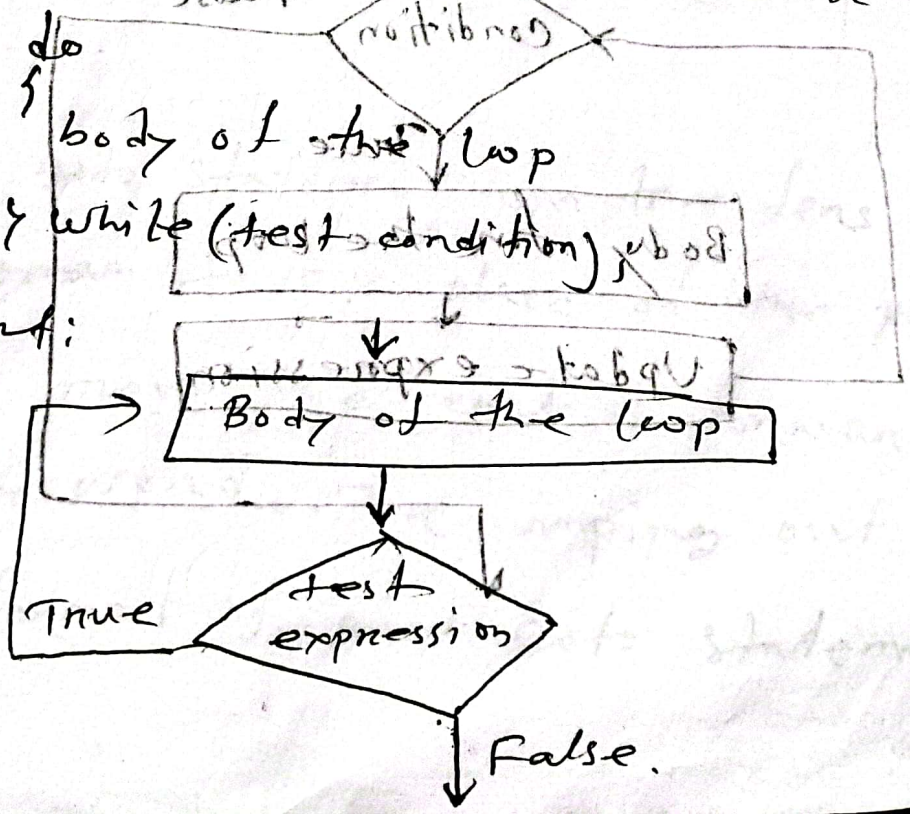
While loop body contain statements.

Do while:

do-while loop is similar to while loops, but it always execute the code block at least once as long as the condition remains true. syntax of do-while loop is

```
do
{
  body of the loop
} while (test condition);
```

flowchart:

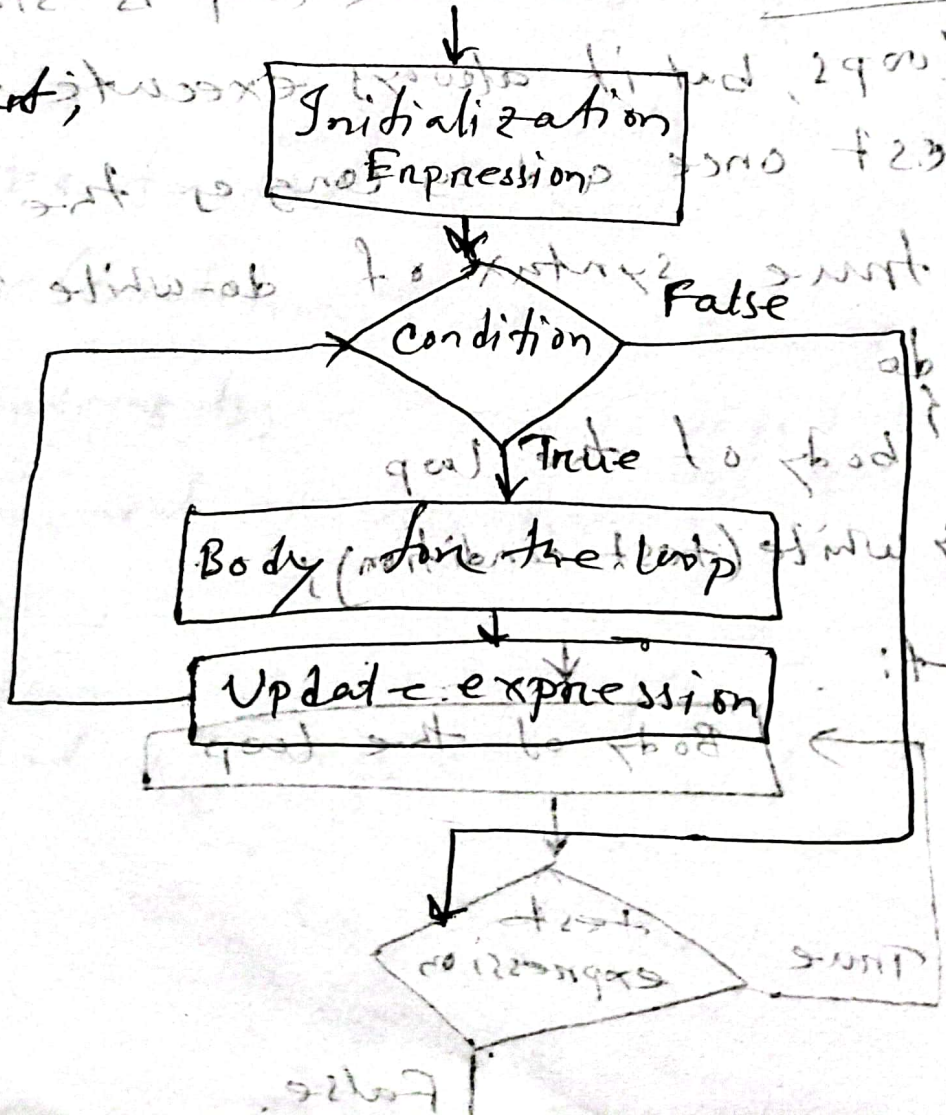


For ~~loop~~ loops C for loops is very similar to a while loops, in that it continues to process a block of of code until the condition becomes false, and everything is defined in a single line. The for loop is also entry controlled loop.

The basic format of for loop statement is

```
for(initialization; condition; increment or decrement)
{
    body of the loop
}
```

flow chart;



Jumping out of a loop: Using break statement and goto statement, we easily perform this task. When a break statement is encountered inside a loop is easily terminated and the program control resumes at the next statement.

The basic format of break in different loop is.

```
while(....)
{
  ...
  ...
  if(condition)
      break;
  ...
  ...
}
```

exit from a loop

```
do
{
  ...
  ...
  if(condition)
      break;
  ...
  ...
} while(....);
```

exit from a loop

```
for(....)
{
  ...
  ...
  if(condition)
      break;
}
```

Since a goto statement can transfers the control of ~~a program~~ to any place of the program. goto can be move the control forward and also in back word. For jumping out of a loop the basic format of ~~goto~~ goto statement is

```
while(....)
{
  ....
  if(condition)
    goto outside;
  ....
}
outside:
```

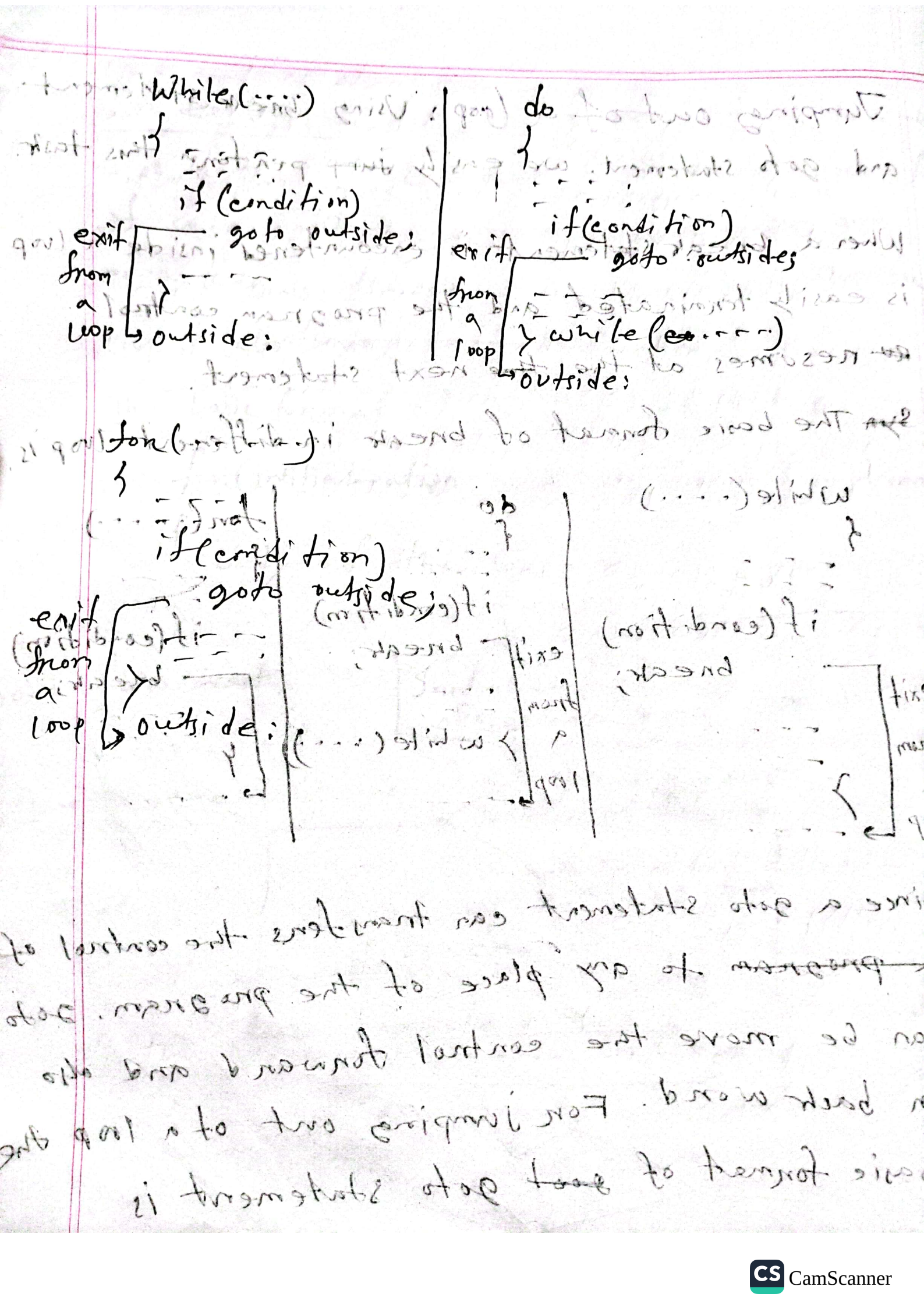
exit from a loop

```
do
{
  ....
  if(condition)
    goto outside;
  ....
} while(....)
outside:
```

exit from a loop

```
for(....;....;....)
{
  ....
  if(condition)
    goto outside;
  ....
}
outside:
```

exit from a loop



Chapter-05 Managing input and output operati

Q Write down the character test functions.

→ isalnum(c) – is c an alphanumeric character?
isalpha(c) – is c an alphabetic character?
isdigit(c) – is c a digit?
islower(c) – is c lowercase letter?
isupper(c) – is c uppercase letter?
ispunct(c) – is c a punctuation mark?
isspace(c) – is c a whitespace character?
isprint(c) – is c a printable character.

Q What is formatted input? Explain

In c programming, scanf() is one of the common used function to take input from the user. The 'scanf()' function reads formatted input such as keyboard.

Formatted input function scanf() is used to take various inputs from the user. This type of function can help to display the output to the user in different formats using the format specifiers. Formatted input function support all data types like int, float, char, double and many more.

For example: 15.75 123

Here, This line contains three pieces of data, arranged in a particular form. The first part of the data should be read into a variable float, The second into int and the

What is the purpose of

third part into char. This is possible in c using the scanf function. scanf means scan formated.
scanf() function is used in c program for reading or taking any value from the keyboard by the user. This function is declared in stdio.h (header file), that's why it is also predefined function. In scanf() function we use address of operator (&) which is used to stores the variable value on the memory location of that variable. The general format of scanf() is.

scanf("Format specifier", &var1, &var2 ... &varn);

or

scanf("control string", &arg1, &arg2, ... &argn);

example:

```
main()
{
    int a;
    float b;
    printf("Enter a integer and a float:\n");
    scanf("%d.%f", &a, &b); //taking multiple input
}
```

This program is for taking two formatted input by scanf.

Some points to remember while using scanf().
some of the general points to keep in mind while writting a scanf() statement:

1. All function arguments, except control string must be pointers to variable

2. Format specifiers must be contained in the control string, should match the argument in order.

3. Input data items must be separated by spaces and must match the variable receiving the input in the same order.

4. The reading will be terminated, when scanf encounters a mismatch of data of a character that is not valid for the value being read.

5. When searching for a value, scanf ignores line boundaries and simply looks for the next appropriate character.

6. When the field width specifier w is used, it should be large enough to contain the input data size.

Rules for scanf

① Each variable to be read must have a filed specification. (format specifier).

② For each format specifier, there must be a variable address of proper type.

③ Any non-whitespace character used in the format specifier must have matching in the user input.

④ The scanf reads until:

- A whitespace character is found in a numeric specification, or
- The maximum number of characters have been read, or
- An error is detected, or
- The end of file is reached.

Formatted output: printf() function is used to printing captions or numerical results. it is therefore necessary for the programmer to give careful consideration and clarity of the output produced by his program, thus printf() is very important. The general form of printf() statement is;

printf("control string", arg1, arg2, ...);

Control string may be contain characters that will printed on the screen as they appear, Format specifications that define the output format for display of each item, Escape sequence characters such as \n, \t and \b

examples of printf statements are.

printf("programming in c");
printf("\n");
printf("%d", n);
printf("sum = %d\n", n);

Enhancing the readability of output.

Readability of output is very important. The correctness depends on the solution procedure, the clarity depends on the way the output is presented. Following are some of the steps we can take to improve the clarity and hence the readability and understandability of outputs.

(i) Provide enough blank space between two numbers
(ii) Introduce appropriate headings and variable names in the output.
(iii) print special messages whenever a special condition

(iv) Introduce blank lines between the important sections of the output.

Examples:

```
printf("a= %d\t b= %d", a, b);
printf("a= %d\n b= %d", a, b);
printf("Code\t Name\t Age\n");
printf("Enter your name\n");
```

Q. How can we use the getchar() function to read multiple-character strings.

This is the program to read multi-character string using getchar()

```
main()
{
    char c;
    while((c=getchar()) != EOF)
    {
        printf("%c\n", c);
    }
}
```

If the input is 'ajeet', then the loop will run for 7 times. getchar() reads character one by one. Mainly getchar() reads next

characters from the standard input's buffer and returns it.

Here, in the program, we receive getchar() returned character by 'c' and getting character by getchar() function until the EOF is occurred.

How can we use the putchar() function to output multicharacter strings.

The putchar() method in C used to output a character. The program to output a strings using putchar() is written bellow

```
main()
{
    char name[] = "sajeeb";
    for (int i=0; name[i] != '\0'; i++)
    {
        putchar(name[i]);
    }
}
```

The output of the program is "sajeeb".

Here, putchar() returns the character written on the stdout as an unsigned char.

This function accepts a mandatory parameter char which is the character to be written to stdout.

Chapter 7 | Remaining part |

Q. What is structured programming?

→ Structure programming is an approach to the design and development of programs. It is a discipline of making of program's logic easy to understand. The following basic three control structure are:

1. sequence structure (straight line)
2. selection structure (branching)
3. Repetation structure (looping)

The use of structured programming techniques helps ensure well-designed programs that are easier to write, read, debug and maintain those are structured. structured programming discourage using jump statements such as goto, break and continue. Write down the different process of looping statements.

In programming, a loop is used to repeat a block of code until the specified condition is met. C programming has three types of loops. and they are for loop, do...while loop and while loop. In general, a looping process would include the following four steps.

1. setting and initializing a conditional variable
2. Execution of the statements in the loop.
3. Testing a specified condition for the execution of the loop.
4. Incrementing or updating the condition variable.

What is null statement? Explain with example:

A null statement is a statement containing only a semicolon. It is used to provide a null operation in situations where the language requires a statement, but program need no work to be done. The null statement consist of a semicolon & a ';'... statements such as do, for, if and while requires that an executable statement appears as the statement body. The null statement satisfy satisfies the syntax requirement in cases that do not need actual statement body.

example: for(int i=0; i<10; i++);

Hence the body statement body, is a null statement, since no further statement are necessary.

Compare of goto and continue:

1. The continue statement is automatically beginning the loop statement when it is encountened in the program. Where as the goto statement is used for repeating some part of the code for a particular condition.

2. The continue statement is usually used within the while loop or do-while loop, or for loop where the goto statement is used in any where of the program source code.

3. The syntax of continue is continue; and the syntax of goto is

```
label:
statements;
goto label;
```

Compare of break and goto:

1. The break is used in terminating loop immediately when it is encountened. where goto statement is used for repeating or skiping some part of the code for a particular condition.

2. The break statement is usually used with the switch statement. But the go to statement is used in anywhere of the source code.

3. The syntax of the break statement is break; and the syntax of the go to statement is

label:
statements;
goto label;

Compare of Break and continue.

1. The break statement is used to exit from the loop. Whereas, the continue statement begins the next iteration of the loop.

2. The break statement usually used with the switch statement and it can also use within the while loop, do while loop or the for loop. But the continue statement is not used in the switch statement but it can be used within the while-loop, do-while loop, or for loop.

3. When a break statement is encountered then the control is exited from the loop constructed immediately. When the continue statement is encountered, then control automatically passes

passed from the beginning of loop statements.
4. Syntax of break is break; and syntax of continue is continue;

Compare of while and for loop:

1. In for loop, initialization may be either in loop statement or outside the loop. In while loop, initialization is always outside the loop.
2. In for loop, once the statement is executed then after-increment is done, But in while loop, increment can be done before after the execution of the statement.
3. For loop is normally used, when the number of iteration is known. while loop is normally used when the number of iteration is unknown.
4. In for loop, condition is a relational expression. But in while loop, condition may be expression or non-zero value.
5. For loop used for where initialization is simple. while loop is used for complex initialization.
6. syntax:

for (initialization ; test condition ; increment or decrement)
{ body of loop
}while (condition) { statement; }

Compare of while and do...while loop:

1. In while loop condition is checked first then statement is executed. on other hand, it do...while loop, statement is executed first at least once then condition is checked.

2. In while loop, if condition is false then statement is executed zero items. In do...while loop, at least once the statement is executed.

3. In while loop, no semicolon at the end of the while, (while(condition) but in do..while loop, semicolon at the end of while. while(condition);

4. In while loop, If there is a single statement bracket are not required. But in do...while loop, bracket are always required.

5. while loop is entry controlled loop where do...while loop is exit controlled loop.

6. The syntax of ~~the~~ while and do...while loop as follow;

```
while (condition)
{
    statements;
}
```

```
do
{
    statements;
} while (conditions);
```

Pointer → structure → Union.

Rules of pointer operation:

① A pointer variable can be assigned the address of another variable

② A pointer variable can be assigned the values of another pointer variable

③ It can be initialized with Null or zero value.

④ A pointer variable can be pre-fixed or post-fixed with ++ increement and decrement operators.

⑤ A integer value may be added or substructed from a pointer variable

⑥ When two pointers point to the same array, one pointer variable can be substructed from another.

⑦ When two pointers point to the object of the ~~array~~ same data types, they can be compared with using relational operators.

⑧ A pointer variable can not be multiplied by a constant.

⑨ Two pointer variable can not be added.

⑩ A value cannot be assigned to an arbitary address.

Rules for pass by pointers:

(i) The types of the actual and formal arguments must be same.

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

(iii) The formal arguments in the function header must be prefixed by the indirection operator *.

(iv) In the prototype, the arguments must be prefixed by the symbol * (asterisk).

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

```
main()
{
    function(actual arguments);
}
function(formal arguments)
{
}
```

Difference between pass by value and pass by pointer.

The technique used to pass data from one function to another is known as parameter passing. ~~Parameter~~

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 can not be changed accidentally

In pass by pointers; (also known as pass by address) The memory addresses of the variables rather than the copies of the 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.

What is pointer

→ A pointer is a variable that stores memory address. Pointers are used to store the addresses of other variables or memory items. pointers are very useful for another type of parameter passing, usually referred to as pass by address. //

pointers are essential for dinamic memory allocation.

Declaring pointers:

Pointer declaration use the asterisk (*) operator.

```
int *p; // declaration of a pointer called p
```

declares the variable p as a pointer variables that point to an integer data type. Remember that the type int refers to the data type of the variable being pointed by p and not the type of the value of the pointer.

```
float *x; /* float pointer */
```

Declares x as a pointer to a floating-point variable.

The declarations cause the compiler to allocate memory locations for the pointer variables p and x.

Sometimes the notation is confusing because asterisk(*) place differently.

```
int *p;
int* p;
int * p;
```

The three following declarations are equivalent. And the variable p as an pointer to an int.

The pointer maybe int, float, char, double type but all the same size as they just store a memory address.

What is union:

→ A union is a special data type available in C that allows to store different data types in the same memory location. We can define a union with many members, but only the member can contain a value at a given time. Union provide an efficient way of using the same memory location for multiple purpose.

What is structure Define:

→ Arrays allow to define type of is a variables that can hold several data items of the same kind. Similarly, structure is another user defined data type in C that allows to combine data items of different kinds.

The general format of a structure definition is as follows:

```
struct tag_name
{
    data_type member1;
    data_type member2;
    ...
};
```

In defining structure we may note the following syntax:

1. The template is terminated with a semicolon.
2. While the entire definition is considered as a statement, each member is declared independently for its name and type in the separate statement inside the template.
3. The tag name can be used to declare structure variables of its type, later in the program.

Arrays Vs structures:

Both of the arrays and structures are classified as structured data types as they provide a mechanism that enable us to access and manipulate data in a relatively easy manner. But they differ in a number of ways which are as follows

1. An array is a collection of related data elements of same type. Structure can have elements of different types.

2. An array is a derived data type whereas a structure is a programmer defined one.

3. Any array behaves like a build in data type. All we have to declare an array variable and use it But in the case of structure, first we have to design and declare a data structure before the variables of that data type are declared and used.

Declaring structure Variables:

After defining a structure format we can declare variable of that type. A structure variable declaration is similar to declaration of variables of any other data types. steps are

① The keyword struct
② The structure tag name.
③ List of variable names separated by commas.
④ A terminating semicolon

For example the statement

```
struct book_bank book1, book2, book3;
```

declares book1, book2 and book3 as variables of type struct book_bank.

Each one of these variables has same members specified by the template. The complete declaration might look like this,

```
struct book_bank
{
    char title[20];
    char author[];
    int  page
    float price
    ---- ----
    ---- ---- ----
};
struct book_bank book1, book2, book3;
```

The members of a structure themselves are not variables. They do not occupy any memory until they are associated with the structure variable such as book1. When the compiler comes across a declaration statement, it reserves space for the structure variables. It is also allowed to combine both the structure definition and variables declaration in one statement.

The declaration,

```
struct book_bank
{
    char title[];
    char author[];
    int page;
    float price;
} book1, book2, book3;
```

is valid.

Rules for initializing structures: There are a few rules to keep in mind while initializing structure variables at compile time which are follows,

(i) We cannot initialize individual members inside the structure template.

(ii) The order of values enclosed in braces must match the order of members in structure definition.

(iii) It is permitted to have a partial initialization We can initialize only the first few members and leave the remaining blank. The uninitialized members should be only at the end of the list.

(iv) The uninitialized members will be assign default values as follows,

- Zero for integer and floating point numbers.
- '\0' for characters and strings.

Structure Versus union

The keyword struct is used to define a structure
On the other hand, the keyword union is used to define a union.

The size of the structure is greater than or equal to the sum of sizes of its member, whereas the size of union is equal to the size of largest member.

Each member within a structure is assigned unique storage area of location. But memory allocated is shared by individual members of union.

Alternating the value of a member will not affect other members of the structures, whereas Altering the value of any of the member will alter other member values.

In structure, individual member can be accessed at a time, and in union, only one member can be accessed at a time.

Several members of a structure can initialize at once but only the first member of a union can be initialized

Structure elements :→ First define the structure

① The keyword struct

② The structure tag name

③ List of variable names separated by commas.

④ A terminating semicolon

The general form of "fprintf" is

fprintf(fp, "control string", list);

Where fp is a pointer associated with a file that has been opened for writting. The control string contains output specifications for the items in the list. The list may contain/include variables, constants and strings. Example: fprintf(f1, "%s %d %f", name, age, 7.5);

Here name is an array variable of type char and age is an int variable

The general form of ~~"fprintf" is~~ "fscanf" is,

fscanf(fp, "control string", list);

The statement would cause the reading of the items in the list from the file specified by fp, according to the specifications contained in the control string.

Example: fscanf(f2, "%s %d", item, &quantity);

File Management

fprintf: The fprintf function allows the programmer to redirect the standard output to a file instead of the console. Simply put, it lets the programmer write a string into a file.

The syntax or general form of fprintf in C language

```
int fprintf(FILE *fptr, const char *str,...);
```

fscanf: The fscanf function reads a set of data values from a file.

The general form of fscanf in C language

```
int fscanf(FILE *stream, const char *format,...);
```

Some library functions of file operation

fopen()	create a new file for use or Opens an existing file for use
fclose()	Closes a file which has been opened for use.
getc()	Reads a character from a file
putc()	writes a character to a file
fprintf()	writes a set of data values to a file.
fscanf()	Reads a set of data values from a file.
getw()	Reads an integer from a file
putw()	writes a integer to a file
fseek()	sets the position to a desired point in the file.

File Management

ftell() Gives the current position in the file (in terms of the bytes from the start)

rewind() sets the position to the beginning of the file.

Q. How to store data in a file in the secondary memory?

→ If we want to store data in a file in the secondary memory, We must specify certain things about the file, to the operating system. They include the following

(i) Filename
(ii) Data structure
(iii) Purpose

Filename is a string of characters that make up a valid filename for the operating system. It may certain two parts - a primary name and an optional period with the extention.

Example: Input.data
Student.dat

Data structure of a file is defined as FILE in the library of standard I/O function definitions. Therefore, all files should be declared as type FILE before they used. FILE is a defined data type.

When we open a file, we must specify what we want to do with the file. For example, we may write data to the file or read the already existing data.

Following is the general format for declaring and opening a file.

```
FILE *fp;
fp = fopen("filename", "mode");
```

The first statement declares the variable fp as a pointer to the data type FILE. And we know FILE is a structure that is defined in the I/O library. The second statement opens the file named filename and assigns an identifier to the FILE type pointer fp.

The second statement also specifies the purpose of opening this file. Mode can does this job.
Mode can be one of the following:

r open the file for reading only
w open the file for writting only
a open the file for appending (or adding) data to it.

Note that both the filename and mode are specified as strings. They should be enclosed in double quotation marks.

Read write and append mode in Brief:

→ Mode specifies the purpose of opening file.

r open the file for reading only
w open the file for writting only
a open the file for appending.

Write mode: When the mode is writting a file with specified name is created if the file does not exist. The contents are deleted, if the file already exists.

appending mode: When the purpose is appending, the file is opened with the current contents safe. A file with the specified name is created if the file does not exist.

Reading mode: If the purpose is reading, and if it exists, then the file is ~~open~~ opened with the current contents safe otherwise an error occurs.

consider the following statements:

```
FILE *P1, *P2, *P3;
P1 = fopen("data", "r"); // for reading
P2 = fopen("results", "w"); // for writing
P3 = fopen("shedule", "a"); // for appending
```

Error handling within I/O (input/output) operation:
It is possible that an error may occur during I/O operations on a file. Typical error situations include the following.

① Trying to read beyond the end-of-file mark.

② Device overflow.

③ Trying to use a file that has not been opene

④ Trying to perform an operation on a file, when the file is opened for another type of operation.

⑤ Opening a file with an invalid filename.

⑥ Attempting to write to a write protected file.

If we fail to check such read and write errors, a program may behave abnormally when an error occurs. An unchecked error may result in a premature termination of the program or incorrect output. Fortunately, we have two status-inquiry library functions; "feof" and "ferror" that can help us detect I/O errors in the files.

The "feof" function can be used to test for an end of file condition.

If fp is a pointer to file that has just been opened for reading, then the statement,

```
if (feof(fp))
    printf("End of data.\n");
```

would display the message "End of data" on reaching the end of file condition.

The "ferror" function reports the status of the file indicated. The statement

```
if(ferror(fp) != 0) // if(ferror(fp))
    printf("An error has occured.\n");
```

would print the error message if the reading is not successful.

We know that, whenever a file is opened using fopen function, a file pointer is returned. If the file cannot be opened for same reason, then the function returns a NULL pointer.

```
if(fp == NULL)
    printf("File could not be opened.\n");
```

Chapter-3 Constants, Variables and Data Types.

Data Types: Each variable in C has an associated data type. Each data type requires different amounts of memory and has some specific operations. ANSI C support three classes of data type.

(i) Primary (or fundamental) data types.
(ii) Derived data types
(iii) User defined data types

following five primary data types which are commonly used in C.

int: As the name suggests, an integer variable is used to store an integer. Integers are whole numbers with a range of values supported by a particular machine. Range of int in 32 bit is -2^{31} to $+2^{31}$. int usually stored in 2 ~~bites~~ bytes on internal storage.

C has three classes of integer storage, namely short int, int, long int in both signed and unsigned form

Char: The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers. Characters are usually stored in 1 bytes on internal storage. Char may be signed or unsigned.

float: It is used to store decimal numbers (numbers with floating point value) with single precision. float are usually stored in 4 bytes on internal storage. Range of float is ~~1.2 x 10^-38 to 3.4 x 10^38~~ 1.2E-38 to 3.4 E +38

double: It is used to store decimal numbers (numbers with floating point value) with double precision. double are usually stored in 8 bytes on internal storage. double can be long double. Range of double is ~~2.2 x 10^-308 to 1.7 x 10^308~~ 2.2E-308 to 1.7E+308

Void: Void is empty data type that has no value. We use void data type in functions when we don't want to return any value to the calling function.

1) Character set: The characters set that can be used to form words, numbers and expressions depends upon the computer on which program is run. The characters in C are grouped into the following categories:

1. Letters
2. Digits.
3. special characters
4. White spaces.

The compiler ignore white spaces unless they are a part of string constant. White spaces may be used to separate words, but are prohibited between the characters of keywords and identifiers.

Example: - uppercase A...Z
- lowercase a...z
- All decimal digits 0...9
- comma (,), period (.) etc

Page-40

Table-3.1

C Tokens: In a passage of text, individual words and punctuation marks are called tokens. Similarly in a C program individual units are known as C tokens. C has six types of tokens

keywords: Every C word is ~~also~~ classified as either a keyword or an identifier. All key word has fixed meaning and that is can't be changed. keywords serve as basic building blocks for program statement. All keyword must be written in lowercase

Example: auto, double, int, return etc. Page-42 Table-3.3

Identifiers: Identifiers refers to the names of variables, functions and arrays. This are user defined names and consist of a sequence of letters and digits, with a letter as a first character. Both uppercase and lowercase letters are permitted although lowercase letters are commonly used.

The underscore character is also permitted in identifiers. It is usually used as a link between two words in long identifiers.

Constants: Constants in C refer to fixed values that do not change during the execution of a program. Constants may be numeric constants or character constants. Constants may belong to any of the data types.

String: The string can be defined as the one-dimensional array of character terminated by a null ('\0'). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupied one byte of memory, and last character always be '\0'.

Operators: operators are symbols that help in performing operations of mathmatical and logical nature. C operators are, Arithmatic (+, -, *, /, %), Relational (==, !=, <, <=, >, >=), Logical (&&, ||, !), Assignment (+=, -=, *=, /=, %=), Increment and decrement (++, --), Conditional (? :),

Bitwise operators (&, |, ^, <<, >>), special operator (?, sizeof),

special symbols: ~~I c programming language~~

The symbol that are used in C/C++ with some special meaning and some specific function are called as special symbols.

Examples: Brackets [ ], Braces { }, Comma (,), semicolon (;), Parenthesis ( ), Asterisk (*), Assignment operator (=), Preprocesson (#).

Rules for identifiers:

① First character must be an alphabet (or underscore)
② Must consist of only letters, digits or underscore.
③ Only first 31 characters are significant
④ Cannot use a keyword.
⑤ Must not contain white space.

Constants: constants in C refer to fixed values that do not change during the execution of a program. C supports several types of constant.

Integer constants: An integer constant refers to a sequence of digits. It can be an octal integer or a decimal integer or even a hexadecimal integer. The integer constant used in a program can also be of an unsigned type or a long type.

Ex: Decimal integer constant → 55

Hexa decimal integer constant → 5AB

Octal integer constant → 023

Real constants: also known as floating point constants. This type of constant must contain both the parts decimal as well as integers. These quantities are represented by numbers containing fractional parts like 17.548. Sometimes, the floating-point constant may also contain the exponential part.

Example,

We represent the floating point value 3.14 as 3E-14 in the exponent form.

Single character constants: A single character constants (or simply character constant) contains a single character enclosed within a pair of single quote mark.

Example: '5' 'x' ';' ' '

'a' = 97, 'z' = 122

String constants: The string constants are collection of various special symbols, digits, characters and escape sequences that get enclosed in double quotations.

The definition of a string constant occurs in a single line. "This is cookie", "Hello" are the example:

Variables: A variable is a data name that may be used to store a data value. A variable may take different values at a different times during executio

Variable declaration:

① Variable name must begin with letter or underscore.

② Variables are case sensitive.

③ They can be constructed with digits, letters.

(iv) No special symbols are allowed other than underscore.

(v) Sum, height, — Value — are some examples for variable name.

Overflow and underflow of data: Overflow and underflow happen when we assign a value that out of range of the declared data type of the variable. If the value is too big, we call it overflow, If the value is too small, we call it underflow.

The largest value that a variable can hold also depend on machine. C does not provide any warning of integer overflow. It simply gives incorrect result.

Defining symbolic constant: We can use unique constants in a program. Before defining a constant we must ensure (i) modifiability and understandability. Because, modification and some time need to modify these constants and obviously the (ii)

purpose of constant is understable by the readers of the program. That's why the name of the the constant is important. A constant defined as follows;

#define symbolic_name value of constant.

Note that, ① '#' must be the first character of the line and the word 'define' must be written in lowercase. No space between '#' and the word define.

② Symbolic names have the same form as variable names.

③ There is no semicolon after the statement.

Example: #define pi 3.1416

#define pass_marks 33

Variable: A variable is a data name that may be used to store a data value. A variable may take different values in different times during execution.

A variable name can be chosen by the programmer in a meaning full way. Some examples of such names are. Average, height, total. etc.

Reading data from keyboard: In c programming, scanf is one of the commonly used function to take input from the user. The scanf() function reads formatted input from the standard input such as keyboards. The general format of scanf is as follows:

scanf("control string", &variable1, &variable2 ...);

Here, the control string contains the format of data being received. The ampersand (&) symbol before each variable name is an operation that specifies the variable names address. We must always use this operation. for example.

scanf("%d %d", &marks, &marks2);

Since the control string "%d" specifies that an integer value is to be read from the keyboard, we have to type the value in integer form.

Meaning of value of a variable: C variables is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable. The value of the C variable may get change in the program. C variable might be belonging to any of the data type like, int, float, char double etc.

Difference between variables and symbolic names:

symbolic name: A symbolic name is a constant that given to some numeric constant or a character constant or a string constant or any other constant. #define is used for defining symbolic constant.

Variable: A variable is a data name that may be used to store a data value. Variable names may be consists of letters, digits, and underscore (_) character.

Declaration of a symbolic name just defines a name that can be used in a program. Since the the symbolic names are constant. their value can't be changed in the program. On the other hand variable value can be changed within the program.

Derived data type: Derived data type are those that are defined in terms of other data types. Array, functions, pointers, ~~structure and union~~ are derived data type.

User defined data type: Those data types which are defined by the user, structures, union, ~~typedef~~ typedef, enum, class are the user defined data types.

Chapter-4

Operators and Expressions

What is operator in C? Briefly describe the types of operator.

operator: An operator is a symbol that tells the compiler to perform specific mathmatical or logical functions. Operators are used in programs to manipulate data and variable.

C operators can be classified into a number of categories.

They include:

(i) Arithmatic operators.
(ii) Relational operators
(iii) Logical operators
(iv) Assignment operators
(v) Increment or decrement operators.
(vi) Conditional operators.
(vii) Bitwise operators
(viii) special operators

Arithmatic operators: C provides all the basic arithmatic operators.

Operation — meaning

'+' — Addition on unary plus

'-' — substruction on unary minus.

'*' — Multiplication

'/' — Division

'%' — Modulo division

Modulo ~~dis~~ division can't be used in floating point data.

Example: a-b, a+b, a*b,

Here a and b are variables and they are known as operands

Arithmatic operation can be 3 types,

① Integer arithmatic (both operands integer)

② Floating point ~~arithmatic~~ arithmatic (both operands floating point)

③ Mixed mode arithmatic.

Relational operators: C supports six relational operators in all. These are,

'<' — is less than

'<=' — is less than or equal

'>' — is greater than

'>=' — is greater than or equal

'!=' — is not equal to

'==' — is equal to

Examples: (4.5 <= 10) TRUE

(4.5 < -10) False. etc.

(a+b) != (c+d) True [only if the sum of a and b is equal to the sum of c and d]

* Logical operations: There are three logical operations in C language. They are logical AND (&&), logical OR (||) and logical NOT (!)

'&&' - (x>5) && (y<5)

It returns true when both conditions true

'||' - (x>10) || (y>=10)

It returns true when at least one condition is true

'!' - !((x>5) && (y>5))

It is reverse the state of the operand "((x>5) && (y>5))" is true, logical NOT operator make it false.

Assignment operator: Assignment operators are used to assign the result of an expression to a variable. The assignment statement

V op= expression;

is equal to, V = V op expression;

Some of the commonly used shorthand assignment operators are:

a += 1 ——— a = a+1

a -= 1 ——— a = a-1

a *= 10 ——— a = a*10;

a /= 2 ——— a = a/2

a %= 3 ——— a = a%3

Increament and decrement operators: C allows two very useful operators and these are the increment and decrement operators. → ++, --

It can be suffix or prefix. For suffix (m++) and prefix (++m). Here 'm' is a variable.

Conditional operators: The conditional statements are the decision making statements which depends upon the output of the expression.

For conditional statements, there is two conditional statement operators and they are, ? , :

Statement,

expression 1 ? expression 2 : expression 3;

Example: a = 10
b = 15
x = (a > b) ? a : b; // output x = b;

In this example, this can be achived using the if-else statements are follows;

if (a > b)
    x = a;
else
    x = b;

Conditional operators are also known as ternary operator.

Bitwise operator: C has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level. These operators are used for testing the bits or shifting them right or left. Bitwise operators may not be applied to float or double.

& - Bitwise AND
| - Bitwise OR
^ - Bitwise exclusive OR
<< - shift left
>> - shift right.

Special operators: C supports some special operators such as comma operator (,), sizeof operator, pointer operators (* , &) and member selection operator (.).

Expressions: An expression is a combination of operators, constants and variables. An expression may consist of one or more operands, zero or more operators to produce a value.

Arithmatic operator (Integer Arithmetic): When both operands are integers then the result of the arithmatic operation between two integer operands an integer value. Let's take two variables a and b, such that a = 10 and b = 4

now, a + b = 14	a / b = 2
a - b = 6	a % b = 2
a * b = 40	

We know that 10/4 = 2.5 but because both operands are integers, The decimal value is truncated.

Real arithmatic

Floating point arithmatic: An operation between two floating point operands a floating point results. Let's take two variable a and b, such that

a = 11.2 and b = 4.5

Now a+b = 15.7 | a*b = 50.400000

a-b = 6.700000 | a/b = 2.488889

The modulas operator can't be used with floating point constant.

Mixed mode arithmatic: When one of the operands is real or floating point and the other is integer, the expression called mixed mode operation.

In mixed-mode arithmatic expressions, Integer operands are always converted to floating point before carring out any computation. As a result, result of mixed mode operations is real type

Example: 1 + 2.5 = 3.5

Chapter 2 // Overview of C

Why and when do we use the #define directive.

→ In the C programming language, the #define directive allows the definition of macros within our source code. These macro definitions allow constant values to be declared for use in our code.

Macro definitions are not variables and cannot be changed by our program code like variables. We generally use this syntax when creating constants that represent numbers, strings or expressions.

The syntax of using #define in the language is,

#define constant_name value

or

#define constant_name expression

Here, Most C programmers define their constant names in uppercase. Value contain the value of the constant. Here expression means whose value is assigned to the constant. The expression must be enclosed in parantheses if it contains operators.

Example: #define AGE 10 | #define AGE (20/2)

Why and where do we use the #include directive.

In C programming language, the #include directive tells the processor to insert the contents of the another file into the source code. Include directive are typically used to include the c header files for c functions that are held outside of the current source file.

The syntax for the #include directive in the c language is:

```
#include<headerfile>
```

Here, headerfile contains the name of the headerfile that we wish to include. A headerfile is a file that typically ends in ".h".

Example:

```
#include<stdio.h>
int main()
{
    int a=10, b=20;
    int sum=(a+b);
    printf("The result is %d", sum);
    return 0;
}
```

In this program, we are using the #include directive

to include the stdio.h header file, which is required to use the printf standard c library function in our code.

What does void main(void) mean?

Here, the void main() indicates that main() function will not return any value. The void in parathesis (void) means that the main function accepts no arguments.

When our program is simple and it is not going to terminate before reaching the last line of the code or the code is error that we can use the void main().

What do we need to use comments in programs?

In computer programming language, A comment is a programmer readable explanation in the source code of a computer program. They are added with the purpose of making the source code easier for humans to understand. And generally ignored by the compilers and the interpreters.

Comments may be multi line or a single line, depending upon what we need.

Single line comments are started with double slash.

Example:

```
// this is a single line comment
```

it can be anywhere in our program.

Multi line comments are started with slash asterisk (/*) and ends with a asterisk slash (*/). like a single line comment, multi line comment can be write anywhere in the program.

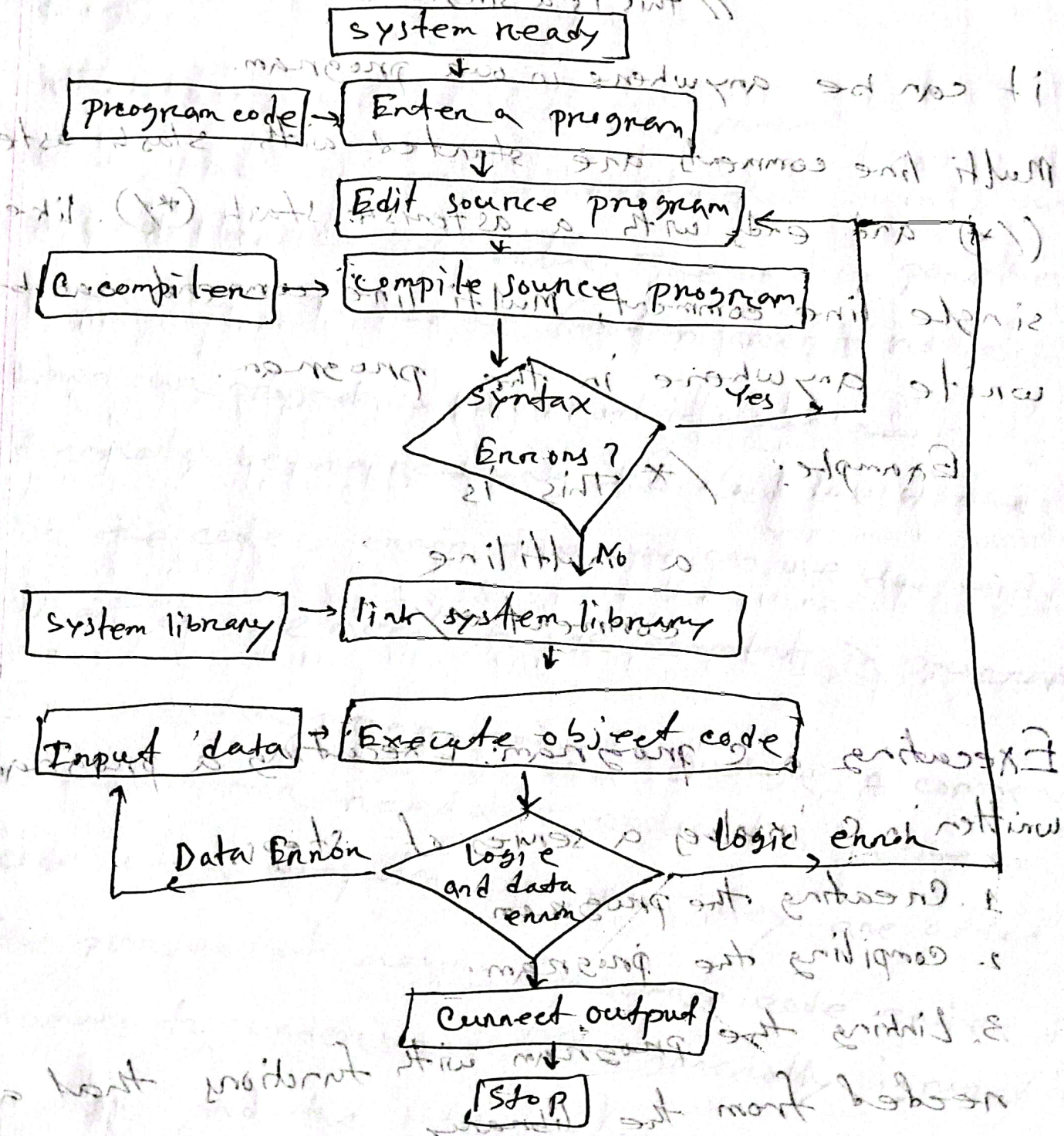
Example:

```
/* This is
a multi line
comment */
```

Executing a C program: Executing a program written in C involves a series of steps:

1. Creating the program.
2. Compiling the program.
3. Linking the program with functions that are needed from the library.
4. Executing the program.

The process of creating, compiling and executing a program illustrates bellow.



Key features of C programming Language:

(i) Easy to write and compilation speed is very fast.

(ii) C is a middle level programming language, it can be used for low or high level programming.

(iii) C has many build in functions.

(iv) C supports dinamic memory allocation.

(v) C has a rich set of build in operators.

(vi) C is a structured programming language. It has main opening and closing braces for block of code.

(vii) The language has been extended by several different language.

Basic Structure of a C program:

Documentation section: The documentation section is a part of the program where the programmer gives the details associated with the program.

Link section: This part of the code declares all the header files that will be used in the program.

Global declaration section: This part of the code is the part where the global variables are declared.

Main function: Every c program needs to have the main function. Each main function contains two parts. A delaration part and an Execution part.

Subprogream Section: All the user defined functions are defined in this section of the program.

Chapter-10
remainig part

Difference between automatic and static variables.

→ Both auto (automatic) and a static variable are local variables.

Static variables can retain the value of the variables between different function calls. But scope of auto variable is within the function only. It can't retain the value of the variable between different function calls.

Static variable syntax,

static data-type variable name = value;

automatic variable syntax:

auto data-type variable_name;

Difference between global and local variable

→ A global variable is a variable that is accessible globally, whereas A local variable is one that is only accessible to the current scope, such as temporary variables used in a single function definition.

Example:

```
int p=10;
main()
{ q=20;
} Here in the program p is global and q is local
```

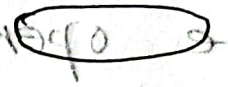
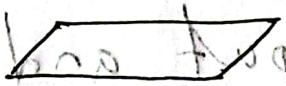
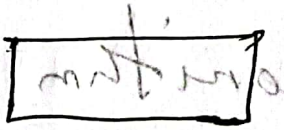
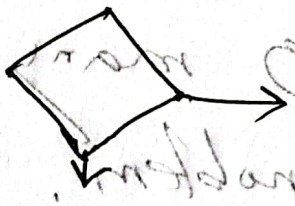
Difference between scope and visibility of a variable.

→ Scope is the region of a program in which a variable is available for use. But visibility is the program's ability to access a variable from the memory.

Visibility is the result of hiding a variable in outer scope. whereas the scope of an identifier is that portion of the program code in which it is visible.

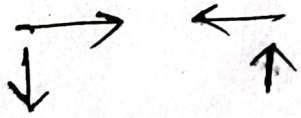
Flowchart: A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagramatic representation of a an algorithm, step by step approach to solve a task.

Notations are,

- Terminal - Symbol 
- Input/output box 
- Process box 
- Decision box 

Connector 

Pass off connector 

- Arrowed lines 

Algorithm: An algorithm is a series of instructions telling a computer how to transform a set of facts (data) into useful information.

Rules or characteristic features are:

(i) The operations must not be changed
(ii) The operations must be effective
(iii) The operations must be finite
(iv) Input and output must be defined exactly
(v) Algorithm should be most effective among many different ways to solve the problem.