C Programming

→ History of C
- Founded on 1972 by Dennis Ritchie at Bell Laboratories (USA)
- First implanted on DEC PDP-11 computer
- C is very flexible and versatile, allowing maximum control with minimal commands.
- High level language
- Structured language
- Rich library - for arithmetic, etc
- Extensible - Programs are extensible
- Recursion - calling the func in between
- Pointers - Directly interact the computer memory
- Faster -
- Memory managment

→ Tokens
* Keywords : variables which have special meaning and are predefined in the libraries
eg: For, if, else, main (32 in total)
* Constants (Literals) : The value of constants are fixed. Once declared, can't be changed
eg: const int a;
const int *a;
- Types of constants
i) Integer
ii) Floating point
iii) character
iv) String
v) Octal & hexadecimal

* Strings : Collection of characters defined in form of an array and end with null character.

eg: char name[length]

* Special Symbols (single / sequence characters): that have a special build in meaning in the language and typically cannot be used in identifiers.

eg: %, &, (), {}, ..

* Identifiers : Names that declared in a program in order to name a value, variable, function, array and etc.

eg: int x=10;

* Operators
  - Arithematic : +, -, x, ÷, %
  - Increment / Decrement : i++, ++i, i--
  - Assignment : =, ==
  - Relational : <, >, =, !=
  - Logical : AND(&&), OR(||), NOT(!)
  - Bitwise : & (bitwise AND)
    | (bitwise OR)
    ~ (bitwise NOT)
    ^ (XOR)
    << (left shift
    >> (right shift)

Datatypes and Variables

- Basic datatype: int, char, float, double
- Derived: array, pointer, structure, union
- Enumeration: enum
- void: void

- Variables are reserved memory space
  - The value is not constant
  - It can be changed
  - 5 types of variables are:

i) local: that is declared at the inside a code block or func and has scope confirmed to that particular block.

```
eg: void aja(){
        int local_variable = 10;
    }
```

ii) Global: declared at the outside a code block or a functions and has scope accross the entire program and allows to change value.

```
eg: int Global_variable = 10;
    void aja()
    {
      int local_variable = 20;
    }
```

iii) Static Variables: declared using the keyword static. They retains the declared value throughout the entire execution of program. the value will not be changed here in between multiple function calls.

```
eg: static int Static_variable = 10;
```

iv) Automatic variable: declared by using the keyword auto. By default all variables declared in C are auto

eg: auto int auto = 20;

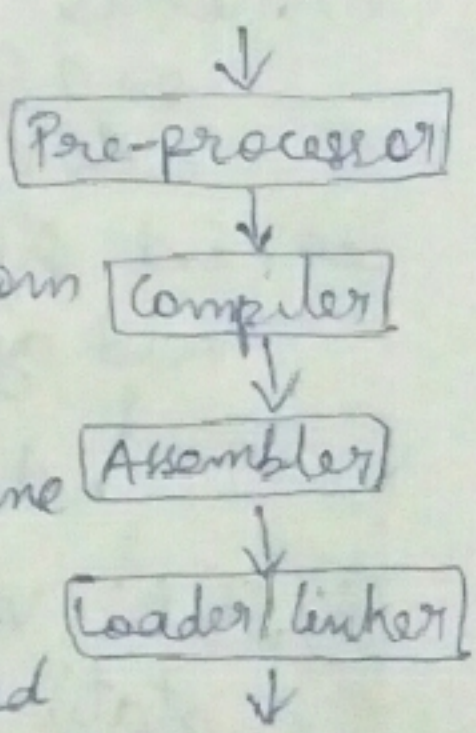
v) External variable: keyword used is extern. We can share a variable in multiple C source files by using an external variable.

eg: extern int external = 10;

* Rules for declaring variables
  - variable can have alphabets, digits and underscore
  - They can start with alphabet and underscore only it can't start with digits
  - No white space allowed
  - Should not be a reserved or keyword

→ Preprocessor Directives

They are lines included in a program that begin with the character '#', which make them different from a typical source code text. They are invoked by compiler to process some program before compilation. It is a ~~many~~ macro processor that is used automatically by the C compiler to transform the program before actual



eg: #include <filename>
#include "filename"

- #define : It is used to define constant or a micro substitution
  eg: define token value
- #undef : Used to undefine the constant or macro defined by #define.
  eg: undef token
- #ifdef : Checks if macro is defined by #define. If yes, it executes the code, otherwise #else code is executed
  eg: #ifdef MACRO
  // code segment
  #endif
- #ifndef : Checks if macro is not defined by #define. If yes, it executes the code otherwise #else code is executed
  eg: #ifndef MACRO
  // code
  #endif
- #if : Evaluates the expression or condition. #elseif or #else or #endif is part of this
  eg: #if expression
  // code
  #endif
- #else : That executes when #if condition fails
- #error : Indicates error. The compiler gives fatal error if they found and skips further compilation process.
- #pragma : Used to provide additional information to compiler.
  eg: #pragma token.

Q. Write a C program to print 'Hello World'.

```
#include <stdio.h>
int main()
  {
   printf ("Hello World");
   return 0;
  }
```

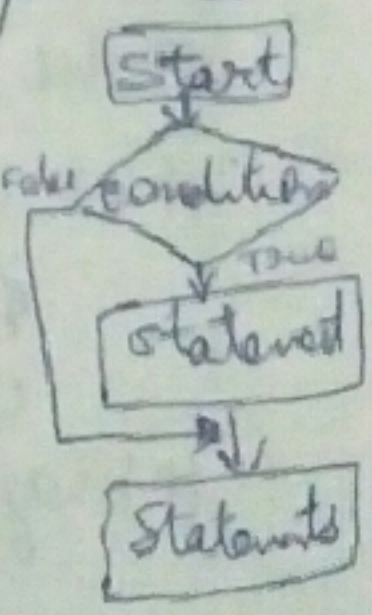
→ Control Statements

They enable us to specify the flow of program control. They specify the order in which the instructions in a program must be executed. They make it possible to make decisions, to perform tasks repeatedly or to jump from one section of code to another.

- if statement
- if-else statement
- if elseif ladder
- Nested if
- Switch
- Ternary
- Break

i) if statement

They are programming conditional statement that, if proved true, perform an operation inside the statement block.

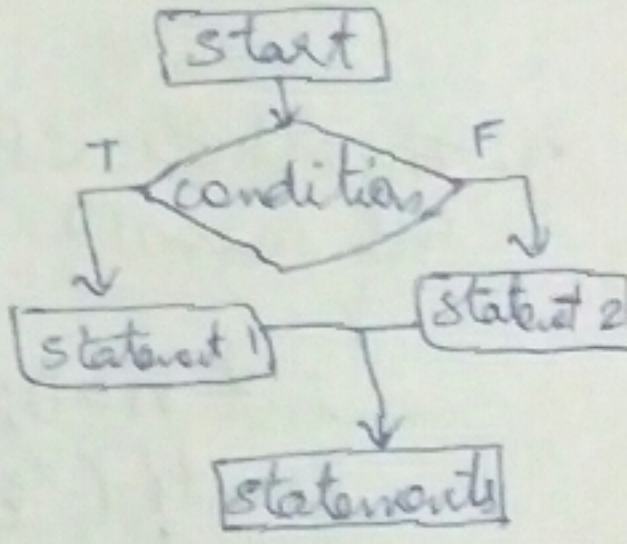


eg:
```
#include <stdio.h>
int main()
  {
   int number;
   printf ("Enter a number");
```

```
    scanf ("%d", &number);
    if (number%2 == 0) {
        printf ("%d is an even", number);
    }
    return 0;
}
```

ii) if-else statement

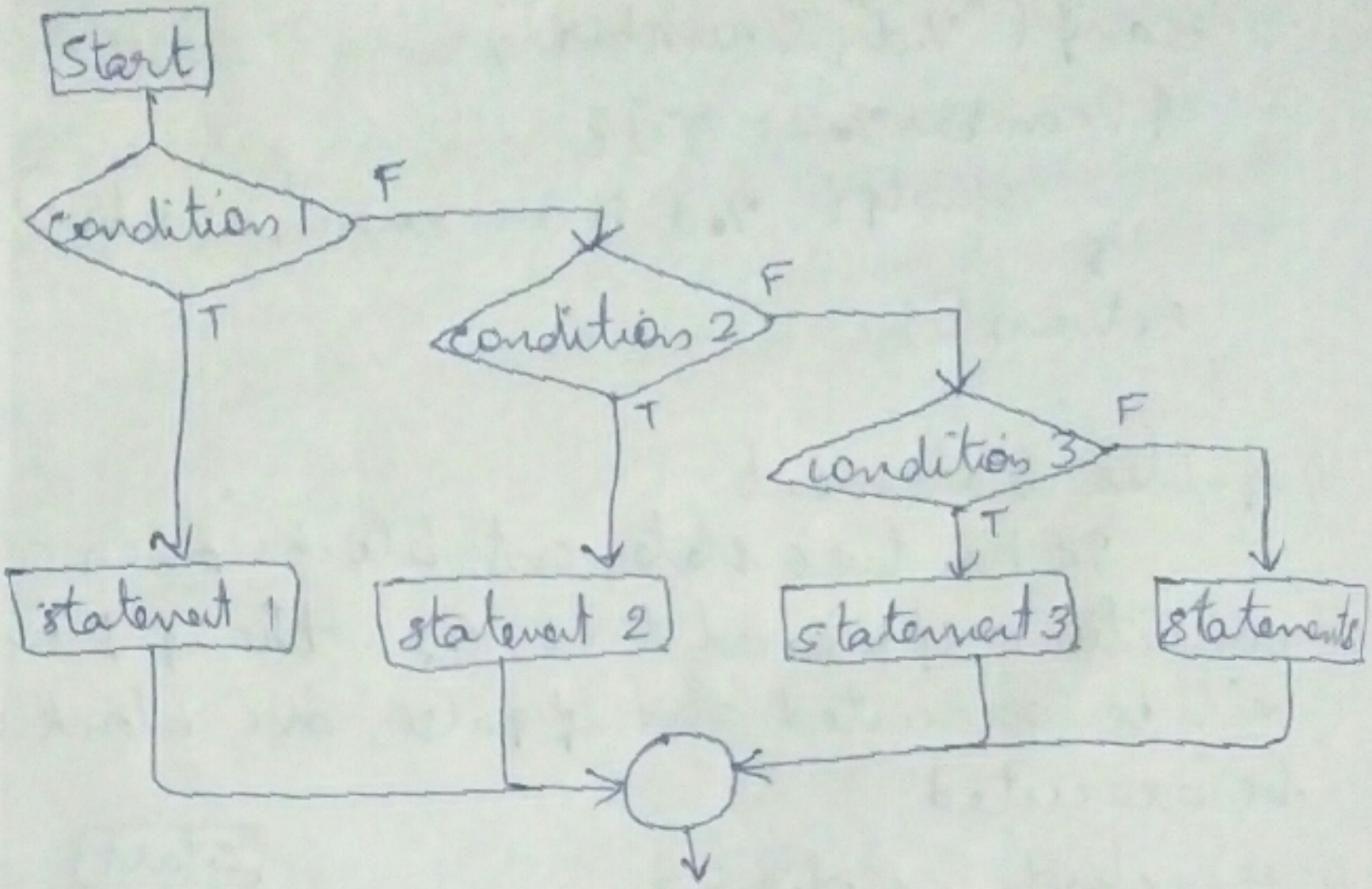
It has two statement blocks over a condition. if proved true, then the if block will be executed and if false, else block will be executed.



eg: #include <stdio.h>
int main()
{
int num = 0;
printf("Enter a number");
scanf("%d", &num);
if (num%2 == 0)
{
printf("number is even");
}
else
{
printf("number is odd");
}
return 0;
}

iii) if else ladder

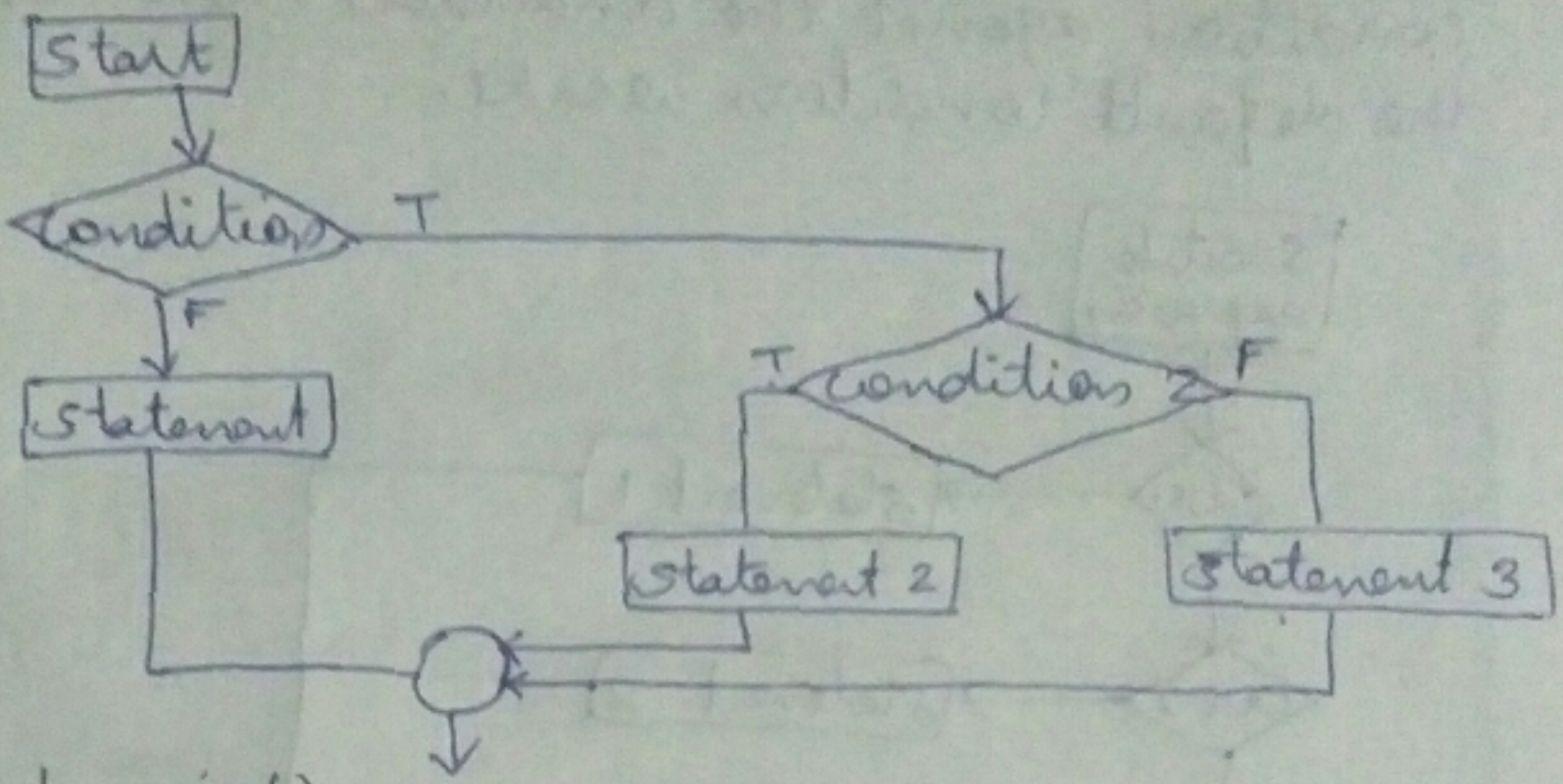
They have multiple else-if statement blocks. If any of the condition is true, then the control will exit the else if ladder and execute the next set of statements.



eg:
```
#include <stdio.h>
int main()
{
 int n=0;
 printf("Enter a number");
 scanf("%d", &n);
 if (n==10)
   {printf("number is 10");
 }
else if(n==5)
    {printf("number is 5");
    }
else if(n==1)
    {printf("number is 1");
    }
else
    {printf("number is not 10,5,1");
}
return 0;
}
```

iv) Nested if conditions.

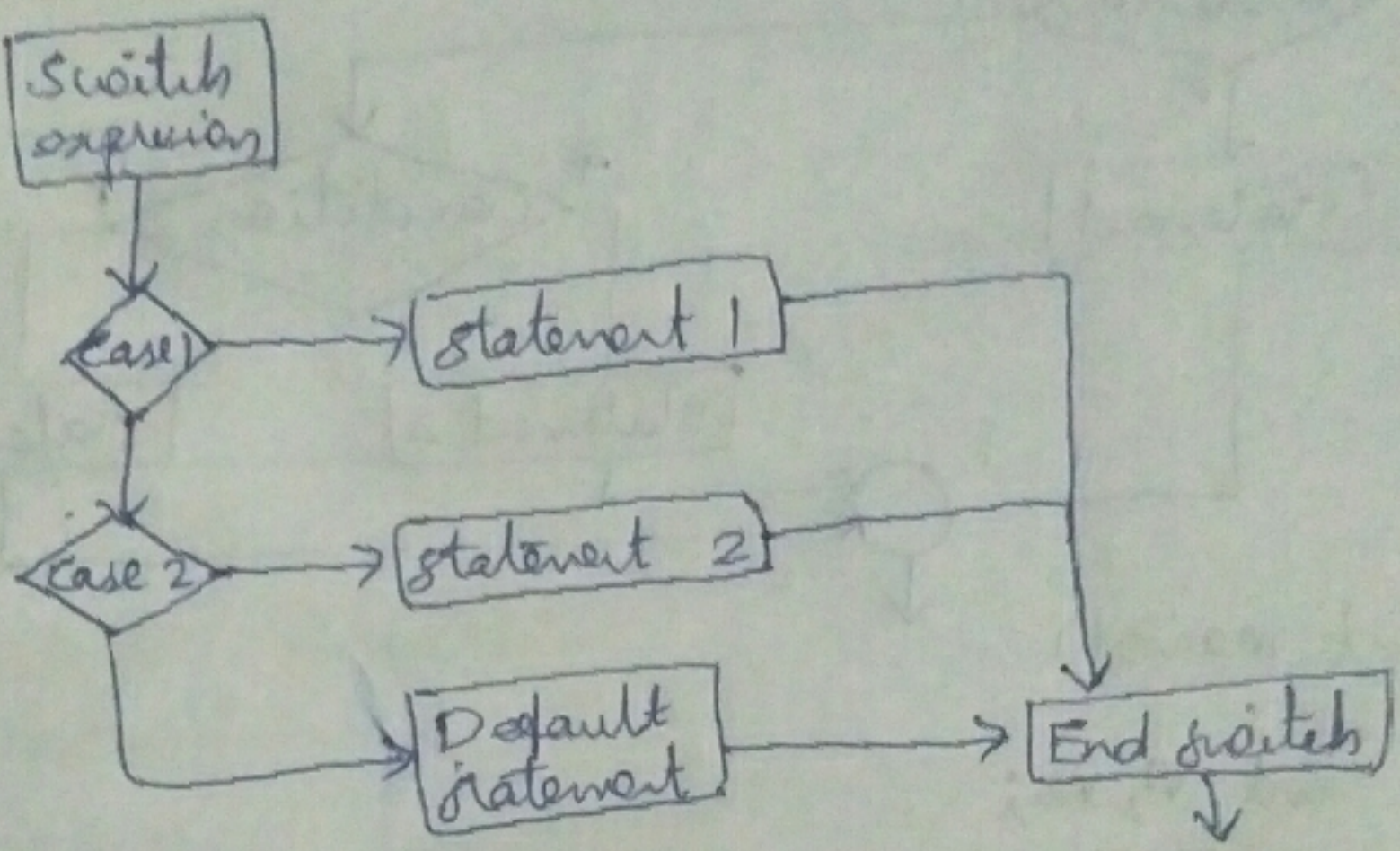
that comprises of another if statement inside the previous if statement.



eg:
```
int main()
{
  int v1, v2;
  printf("Enter first variable \n");
  scanf("%d", &v1);
  printf("Enter second variable \n");
  scanf("%d", &v2);
  if (v1 != v2)
    {printf("%d is not equal to %d", v1, v2);
     if (v1 > v2)
       {
         printf("First is greater");
       }
     else
       {printf("Second is greater");
       }
    }
  else
    {printf("Both are equal");
    }
  return 0;
}
```

v) Switch Statement

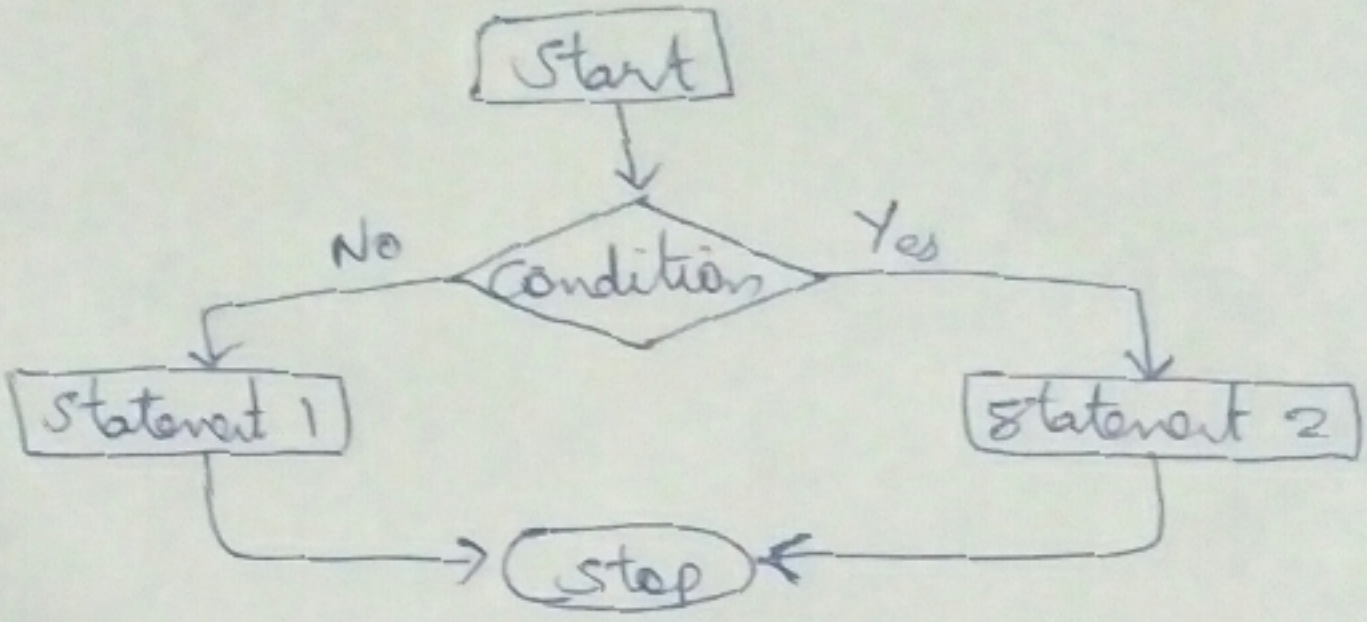
They are programing selection statement that checks for possible match to the provided conditions against the available cases. Otherwise the default condition works.



eg: int main()
{
char op;
double n, m;
printf("Enter ~~operat~~ numbers");
scanf("%.f %.f", &n, &m);
printf("Enter operator");
scanf("%.c", &op);
switch(op)
{
case '+': printf(":%.f", n+m);
break;
case '-': printf("%.f", n-m);
break;
default; printf("Error");
}
return 0;
}

vi) Ternary condition

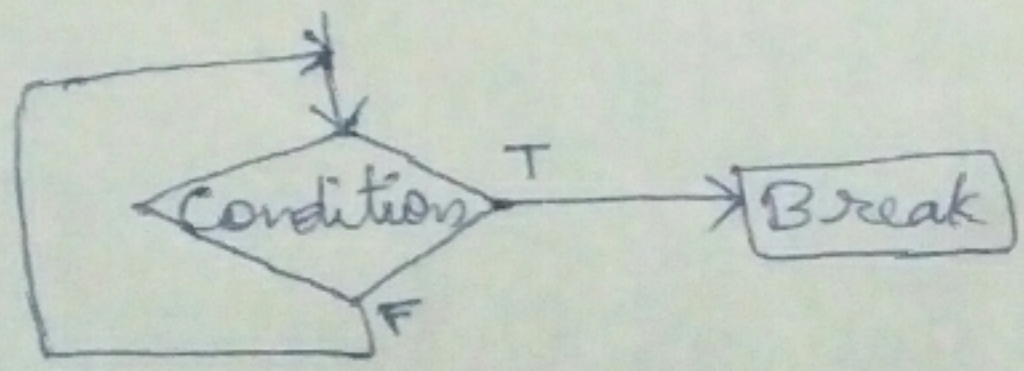
It is similar to an if statement, but shorter in code length. The control checks the condition and executes either of two statements.



eg: int main()
{
int a, b, max;
printf ("%d %d", &a, &b);
max = (a>b) ? a : b;
printf ("%d", max);
printf ("is the largest number");
return 0;
}

vii) Break control statement

They are designed to exit the control from the current code segment to the next code segment when a specific condition is satisfied.

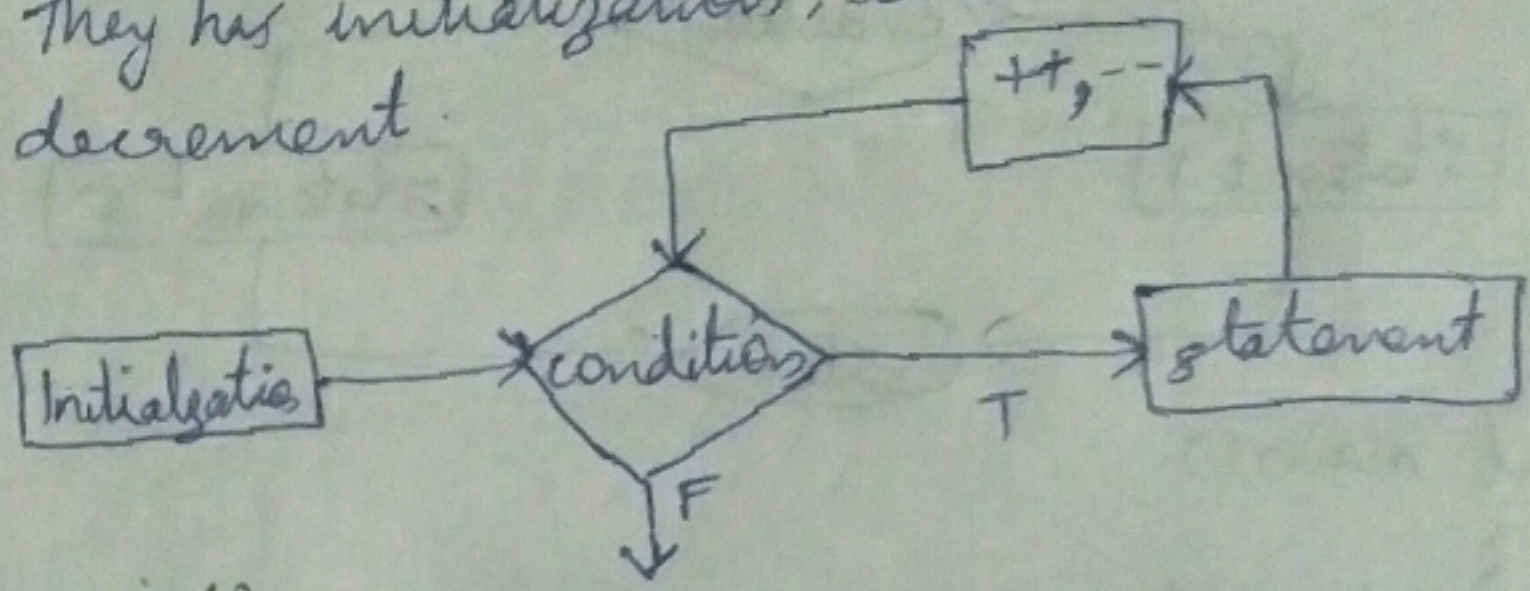


→ Loops

They are used to perform looping operations until the given conditions is true. Control come out of loop statements once it become false.
There are three types of loops:

* For loop

They has initialization, condition and increment or decrement



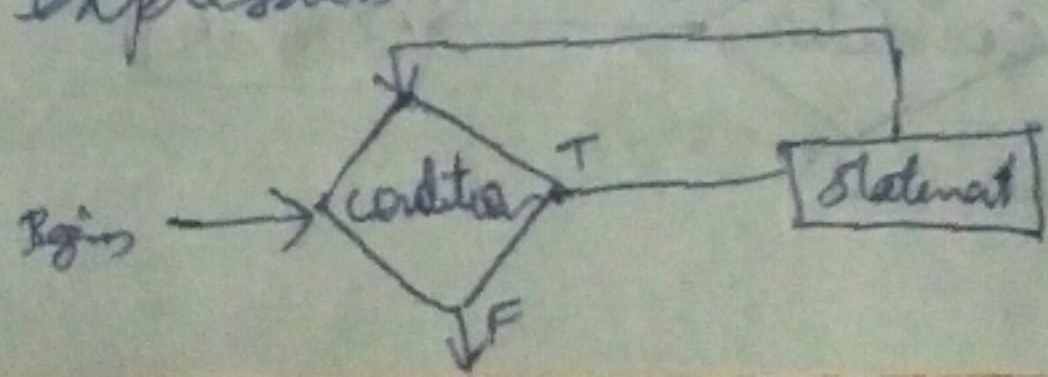
eg: int main()
{
int i;
for(i=1; i<=5; i++)
{
printf("%d", i);
}
return 0;
}

* Nested for loop

Advanced form of for loop which have the for loop inside another for loop

* While loop

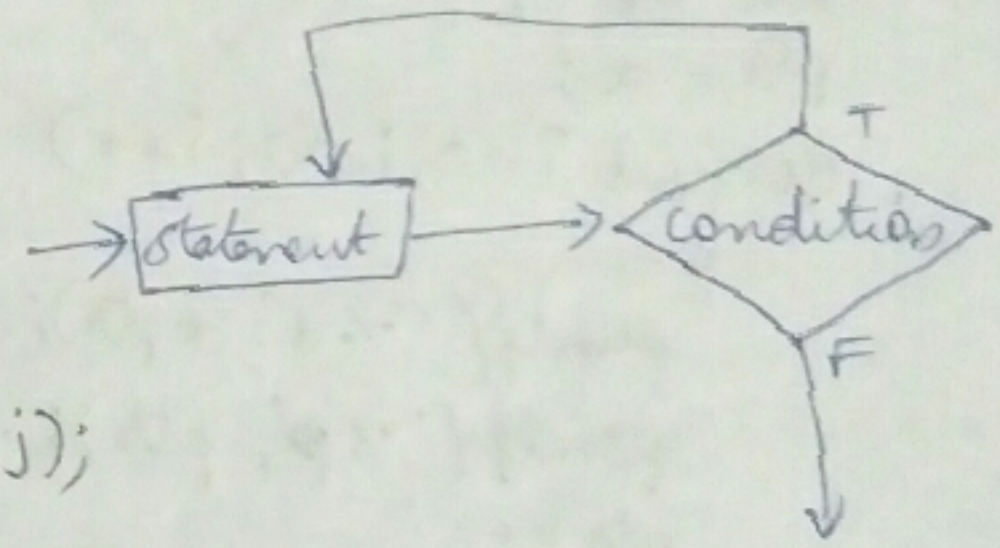
They execute itself repeatedly, until a given boolean expression or a condition is true.



eg:
```
int main()
{
    int count = 1;
    while (count <= 5)
    {
        printf("%d", count);
        count ++;
    }
    return 0;
}
```

* Do while loop

The difference from while loop is that pops up is the condition statement.



g:
```
int main()
{
    int j=0;
    do
    {
        printf("%d", j);
        j++
    } while(j<=5);
    return 0;
}
```

* Infinite loop

They are the loops fails and the execution presists until you stop it manually.

→ Pointers

They are variables which store the address of another variable. The size depends on the architecture. declared as * (asterisk symbol).

eg:
```
int n=10;
int *p=&n;
```

* Advantages of Pointers

- We can return multiple values from a function using the pointer.
- able to access any memory location
- dynamically allocate memory using malloc() & calloc()
- They are widely used in array, func, structure.

eg:
```
int main()
{
    int a[10] = {1,2,3,4,5,6,7,8,9,10};
    int *ptr;
    ptr = a;
    for(int i=0; i<10; i++)
    {
        printf("%d", *ptr);
        ptr++
    }
}
```

→ Escape Sequence

They are a combination of '\' (Backward slash) and a letter or a digit. Used to send non-graphic control characters to specify actions like new line

\a - alarm
\b - Backspace
\f - form feed
\n - new line
\r - carriage return
\t - Tab space

\nnn - Octal number
\v - Vertical tab
\\ - Back slash
\' - Single quote
\" - double quote
\? - Question mark
\xhh - Hexadecimal number
\0 - Null

→ Funtions

They are subdivided program of a main program enclosed within flower brackets. Funtions can be called by the main program to implement its functionality. They provides, code reusability and modularity.

i) Library funtions
ii) User-defined funtions.

* Advantages
- Writting same code repeatedly can be avoided.
- can be called many times
- Easy to understand a program
- Reusability is main.

* Rules for using funtion
- Required to declare as Global and the name, parameter, returntype should be clear.
- While calling funtions, datatype and number of elements in arguments list ~~are~~ have to be matching.
- After declared, func should include parameter declared, code segment and the return value.

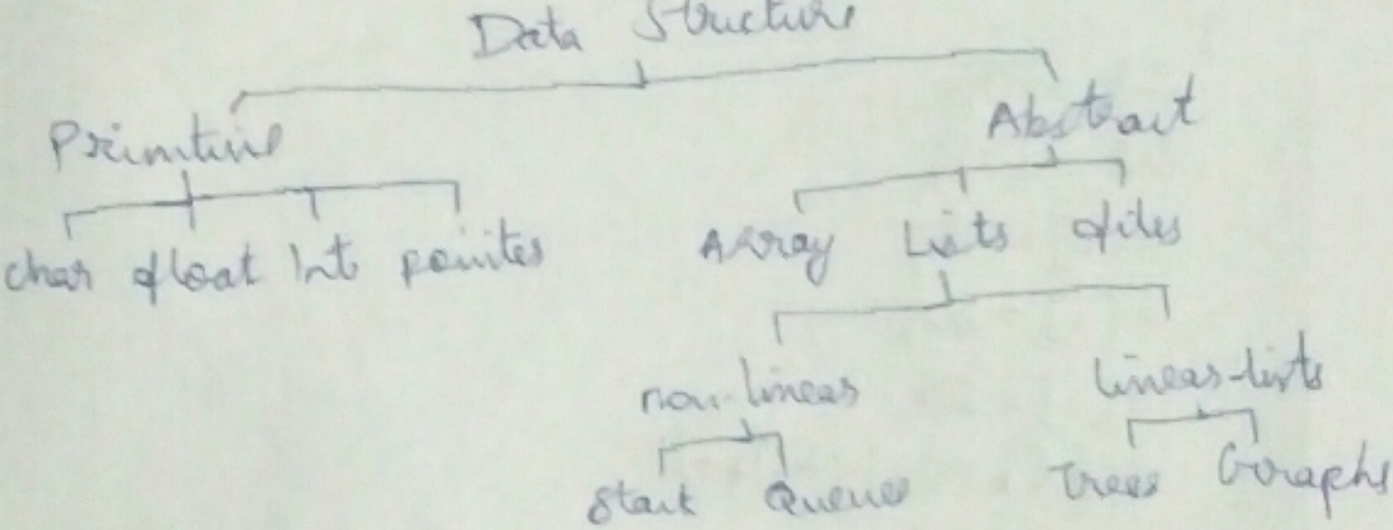
— Aspects of using funtion
- Funtion without argument and without return value.
- Funtion without argument and with return value
- Funtion with argument and without return value.
- Funtion with argument and with return value.

i): void change (int num) {
    printf ("Before %d", num);
    num = num + 100;
    Printf ("After %d", num);
    }
int main () {
    int x = 100;
    change (x);
    return 0;
    }

ii) void swap (int, int);
int main() {
    int a = 10;
    int b = 20;
    Printf("Before %d %d", a, b);
    swap(a,b);
    printf("After %d %d", a, b);
    }
void swap(int a, int b)
    {
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf("After %d %d", a, b);
    }

→ Data Structure
They are collection of data values, the relation among them and functions that are applied on to the data.

- They are:
  - Primitive / Build in
  - Abstract / User defined
- Classification
  - Linear : array
  - Non-linear : Tree, graph
  - Homogeneous : array
  - Non-homogeneous : structure
  - Static : array
  - Dynamic : Linked list

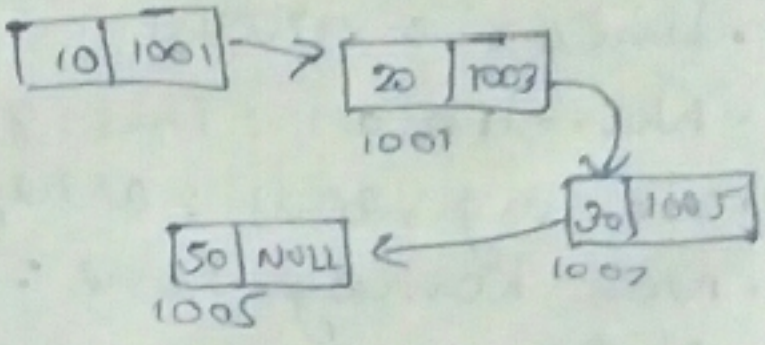


* Arrays
They are collection of similar type of data items stored at contiguous memory location. Each data can be accessed using index number.
  - One dimensional
    arrayRefVar = new datatype[arraysize]; a[]
  - Two dimentional
    ArrayRefVar = new datatype[arraysize]; a[][]
  - Multi dimentional array
    They are the form of matrix which is of the way of 3D.

* Lists

The elements are not saved in sequential memory location, they are stored in random memory addresses and connected with pointers

- Single linked list
- Double linked list
- Circular linked list



Q. One dimentional array

```
int main()
  {
    int i=0;
    int a[3];
     a[0]=10;
     a[1]=20;
     a[2]=30;
     for(i=0; i<4; i++)
        {
          printf("%d", a[i]);
        }
    return 0;
  }
```

Q. Two dimensional

```
int main()
  {
    int a[2][3];
    int i,j;
    for(i=0; i<2; i++)
        {for(j=0; j<3; j++)
            {
              printf("Enter %d %d", i,j);
            }scanf("%d", &a[i][j]);
```

```
printf("two dimensional array");
for(i=0;i<2;i++)
  {
   for(j=0;j<3;j++)
      {
       printf("%d", a[i][j]);
       if(j==2)
          {
           printf("\n");
          }
      }
  }
return 0;
}
```