LECTURE 6

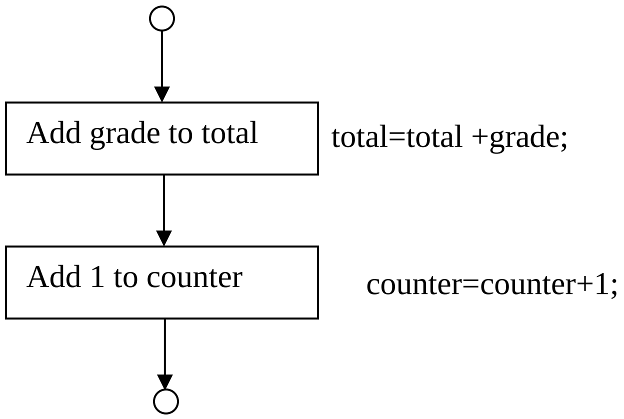
CONTROL STRUCTURES

These are programming statements that control the flow of execution in a program.They are organized into three kinds of control structures.

a. Sequence
b. Selection
c. Repetition/Iteration/Looping

Sequence control structure

The sequence structure is essentially built into C.Unless directed otherwise,the computer automatically executes C statements one after the other in the order in which they are written.



NB:Flowlines indicates order in which actions are performed.

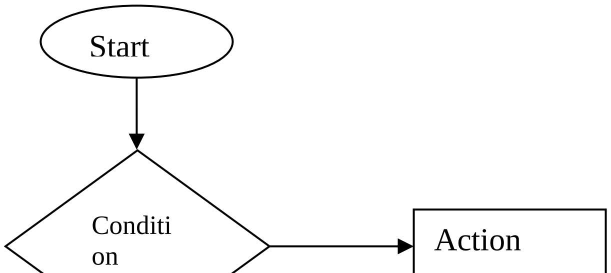
Selection control structure

They are used to choose among alternatives causes of actions.
They include :

i. The if selection statement(Single selection statement)
ii. The if……else selection statement(Double-selection statement)
iii. Switch statement(Multiple-selection statement)

The if selection statement

It performs (selects)an action if a condition is true or skips the action if the condition is false.

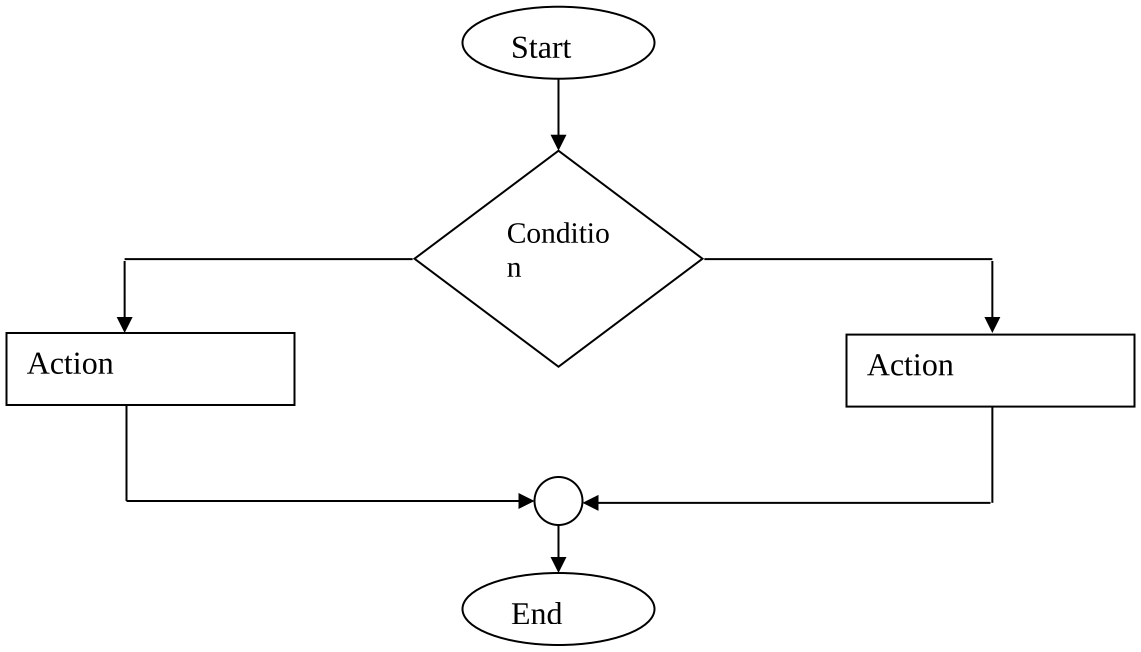


True

```
#include<stdio.h>
int main()
{
int grade;
printf("Enter grade\n");
scanf("%d", &grade);
if(grade>=50)
{
  printf("Passed\n");
 }
  return 0;
}
```

The if……else selection statement

It performs an indicated action only when the condition is true; also condition is for false



```
#include<stdio.h>
int main()
{
 int grade;
printf("Enter
grade\n");
scanf("%d",
&grade);
if(grade>=50)
{
 printf("passed\n");
}
else
{
printf("failed\n");
}
return 0;

}
```

Nested if….else statements
It tests for multiple cases by placing if….else statements inside if….else statements.

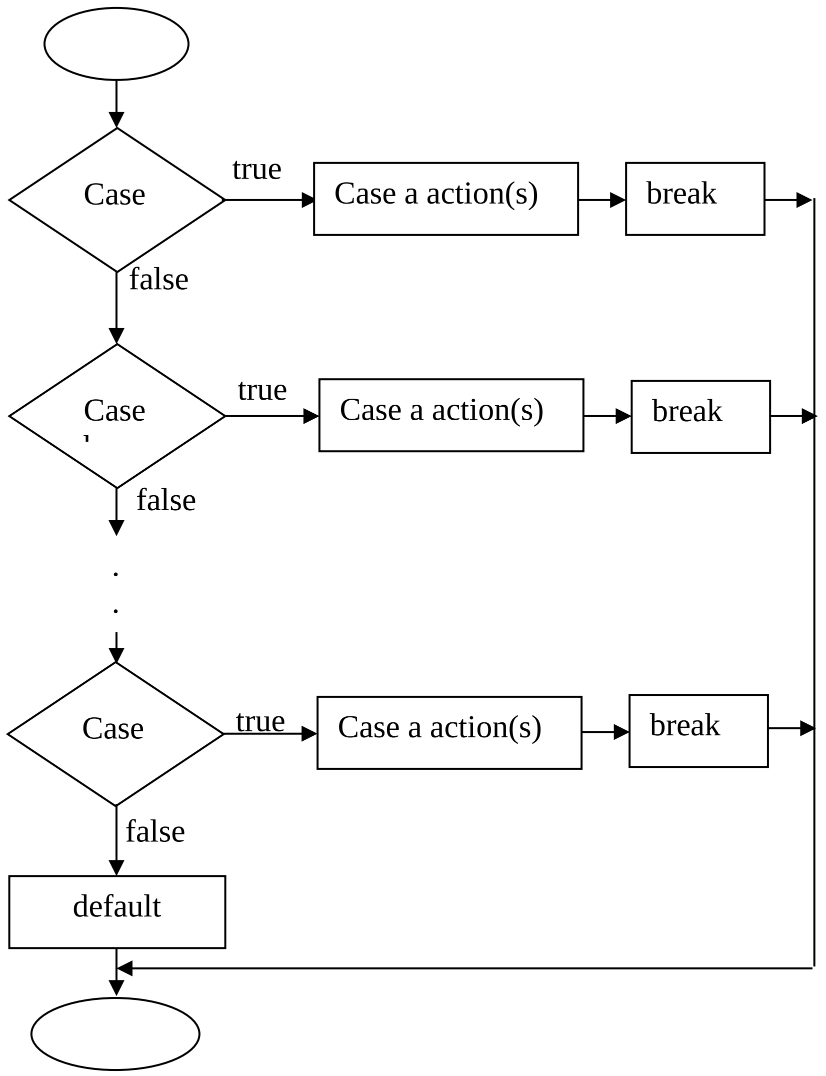
```
#include<stdio.h>



int main()
{
int marks;
printf("Enter grade\n");
scanf("%d", &marks);
if ((marks>=0) &&(marks<=49))
{
 printf("Grade is D\n");
}
else if((marks>=50) &&(marks<=64))
{
 printf("Grade is C\n");
}
else if((marks>=65) &&(marks<=79))
{
 printf("Grade is B\n");
}
else if((marks>=80) &&(marks<=100))
{
 printf("Grade is A\n");
}
else
{
printf("Invalid entry");
}
return 0;
}
```

Switch multiple selection statement
An algorithm contain a series of decisions in which a variable or expression is tested separately for each of the constant integral values it may assume and different actions are taken.This is called multiple selection.
Each case can have one or more actions.
The switch statement consists of a series of case labels,and an optional default case.

Format:

```
Switch(expression)
{
Case (expression1):
```

{
One or more C statements;
}
Case (expression2):
{
One or more C statements;
}
Case (expression3):
{
One or more C statements;
}
…
…
…
Default:
{
One or more C statements;
}
}
If the block is only one statement long,you do not need the braces,but they are recommended.
The default line is optional and it doesn’t have to be the last line of the switch body.



The flowchart makes it clear that each break statement at the end of a case causes control to immediately exit the switch statement.

```
#include<stdio.h>
Int main()
{
Int marks;
Char Grade;
printf("Please enter student marks\n");
scanf("%d", &marks);
switch(marks)
{
 case (marks>=0 && marks<=49)
{
  Grade='D';
  Break;
}
case (marks>=50 && marks<=64)
{
  Grade='C';
  Break;
case (marks>=65 && marks<=79)
{
  Grade='B';
  Break;
case (marks>=80 && marks<=100)
{
  Grade='A';
  Break;
}
default:
{
 Printf("You must enter a number between 0 to 100");
}
return 0;
}
}
```

Repetition/Looping

A loop is a group of instructions the computer executes repeatedly while some loop-continuation condition remains true.
Two main types :

i. Counter-controlled repetition-Also called definite repetition because we know in advance exactly how many times the loop will be executed.e.g for loop
ii. Sentinel-uncontrolled repetition-Also called indefinite repetition because its not known in advance how many times the loop will be executed.e.g do…while and while

Repetition structures include:

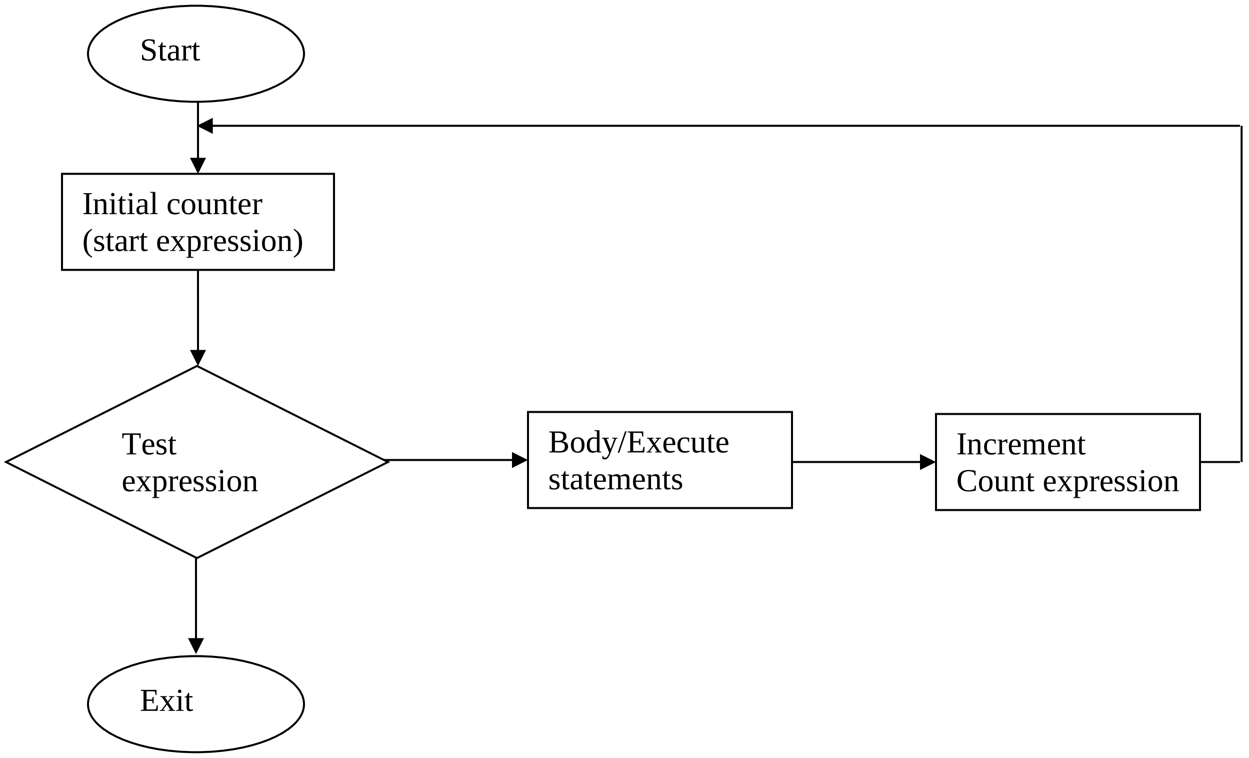
i. while
ii. do------while
iii. for loop

for loop

format of the for loop:

```
for(start expression;test expression;count expression)
{
Block of one or more c statements;
}
```

Start expression is an assignment statement(such as ctr=1;)
Test expression evaluates to true or false,and then determines if the body of the loop repeats again.(loop condition)
Count expression usually increments and decrements.


Start
Initial counter (start expression)
Test expression
Body/Execute statements
Increment Count expression
Exit

Example 1

```
#include<stdio.h>
int main()
{
 int ctr;
for(ctr=1;ctr<=10;ctr++)
{
printf("%d\n",ctr);
}
return 0;
}
```

Example 2

```
#include<stdio.h>
int main()
{
 int total,ctr;
 total=0;
for(ctr=100;ctr<=200;ctr++)
{
total+=ctr;
}
printf("The total is %d\n,total");
return 0;
}
```

Example 3

```
#include<stdio.h>
int main()
{
 int ctr;
 total=0;
for(ctr=10;ctr!=0;ctr--)
{
printf("%d\n,ctr");
}
return 0;
}
```

When decrementing a loop variable,the initial value should be larger than the end value being tested.

output

```
10
9
8
7
6
5
4
3
2
1
```

Nested for loops
When you create a loop within a loop ,you are creating a nested loop.
Example for factorials

```
/*computes the factorial of numbers through the users number*/
#include<stdio.h>
Int main()
{
int outer,num,fact,total;
printf("What factorial do you want to see?");
scanf("%d", &num);
for(outer=1;outer<=num;outer++)
{
total=1;//initialize total for each factorial
for(fact=1;fact<=outer;fact++)
{
total*=fact;
}
}
printf("The factorial for %d is %d",num,total);
return 0;
}
```

output

```
What factorial do you want to see?7
The factorial for 7 is 5040
```

EXAMPLE

Write a program in C to calculate the sum of the first ten (10) positive integers (1-10) using a FOR loop.

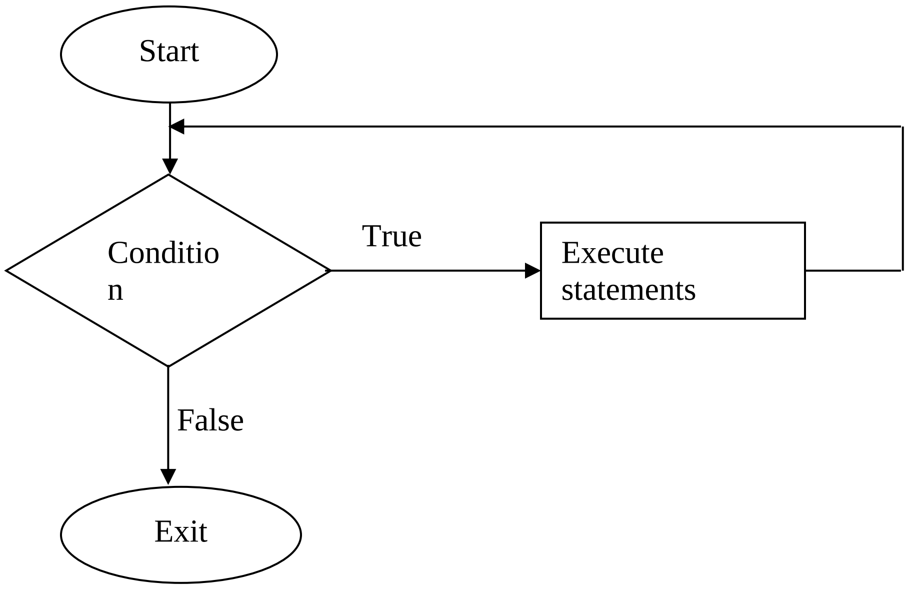
```
#include <stdio.h>
main() {
        int sum = 0;
        int i = 1;
        for (i =1; 1<= 10; i++)
                sum = sum + i;
         printf("The sum is %d\n", sum);
        } // end main
```

While

Format:

```
While(test expression)
{
Block of one or more statements;
}
```

The test expression usually contains relational,and possibly logical operators.
The while loop tests the expression at the top of the loop.



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

```
{
int counter=0;
while(counter<=10)
{
printf("%d",counter);
  counter++;
}
printf("%d",counter);
return 0;
}
```

EXAMPLE

Write a program in C to calculate the sum of the first ten (10) positive integers (1-10) using a while loop.

```
            #include <stdio.h>
            main() {
                    int sum = 0;
                    int i = 1;
                    while (i <= 10)
                    {
                            sum = sum + i;
                            i += 1;
                    }// end while
                     printf("The sum is %d\n", sum);
a.          } // end main
```

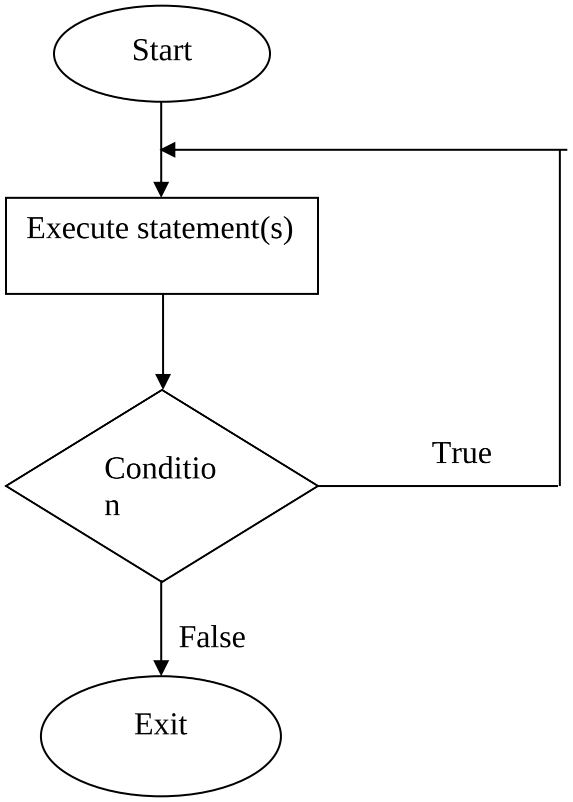
do……while

its similar to the while loop except the relational test occurs at the bottom(rather than top)of the loop.this ensures that the body of the loop executes at least once.

Format:

```
do
{
Block of one or more c statements;
}
while(test expression)
```

it tests the condition after the body is performed.



```
#include<stdio.h>
Int main()
{
int ctr=1;
do
{
printf("%d\n",ctr);
ctr++;
}
while (ctr<=10);
return 0;
}
```

Controlling flow

Using break statement

You use a break statement to quit or exit a loop early.

Using continue statement

It forces the computer to perform another iteration of the loop

Format:

continue;

e.g The first printf() in the for loop executes,but the second does not,due to the continue

```
#include<stdio.h>
main()
{
int ctr;
for(ctr=1;ctr<=10;ctr++)
{
printf("%d",ctr);
continue;//causes body to end early
printf("C programming \n");
}
return 0;
}
```

output

```
1 2 3 4 5 6 7 8 9 10
```

Working with goto statement

It causes your program to jump to different location,rather than execute the next statement in sequence.

format

goto statement_label

statement_label is named just as variables.

If you use a goto statement,there must be a statement_label elsewhere in the program that the goto branches to.

Follow all statement labels with a colon(:)so C knows they are labels and doesn't confuse them with variables

Example

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

```
{
Again:
printf(“this message\n”);
printf(“\t keeps repeating\n);
printf(“\t\t over and over\n”);
goto Again;
return 0;
}
```