Control Structure

Control statements

- This statements are commonly referred to as control statements because they “control” the flow of program execution.
- Control statements change the flow of a program based on specified conditions. There are two types of control statements that is:

  i) Branching control

  - if
  - switch

  ii) looping/iteration control

  - for
  - while
  - do.. while

BRANCHING CONTROL STRUCTURES

IF statement

- The if statement is a powerful decision-making statement and is used to control the flow of execution of statements.
- It is a two-way decision statement and is used in conjunction with an expression, and takes the following form; **if (test expression) statement;**

The if statement may be implemented in different forms depending on the complexity of conditions to be tested as follows:

1. Simple if statement
2. If …. else statement
3. Nested if ….. else statement

Simple if statement

- The general form of a simple ***if*** statement is:

if (test expression) statement;

- The statement block may be a single statement or a group of statements.
- If the test expression is true the statement block is executed; otherwise it will be skipped and the program control jumps to statement .
- E.g.

```
If(avg>50) printf("pass");
```

- Write a program to input voter’s name and age then display “you are allowed to vote” if age is equal or greater than 18.
- Write a program to input student name, maths, english and kiswahili then calculate total and average marks and display the result. If average marks is greater or equal to 50 it should display “pass” else it should display “fail”.

Algorithm

- Input voter name
- Input voter age
- If (age>=18) display “You are allowed to vote”

```
# include <stdio.h>
int main()
{ char vname[10];
    int age;
    printf (“Enter voters’ name”);
    scanf (“%s”,& vname);
    printf (“Enter age”);
    scanf (“%d”,& age);
    if (age>=18) printf (“Vote”);
     else printf (“Not allowed to vote”);
}
```

The If …… Else Statement

- The ***if … else*** statement is an extension of the simple ***if*** statement. Its general form is;

if (test expression) true-statement(s);

else false-statement(s);

- **Example**
- If(avg>50) printf(“pass”);else printf(“fail”);

Example

- Write a program to enter product name, quantity and price then calculate total price. If total price is greater than 10000 a discount of 10% is offered. Otherwise no discount is offered. Let the program display the discount and net price.
- If (tprice>10000) disc=0.1*tprice;
- Else disc=0;
- Netprice=tprice-disc;

- Design a flow chart and a program to input employee name, hours worked and rate per hour then calculate gross pay. Let the program calculate tax 0f 15% when gross pay is greater than or equal to 50000 and 10% when gross pay is less than 50000.

Logical operators

- AND operator (&&) – Applies when both operands/conditions are true.
- OR operator (||) – Applies when both or either condition is true.
- NOT operator (!) – Negates OR or AND operators. E.g. !&&
- If (rain==“yes” && temp<20) prinf (“Wear a Jacket”);

```
if (avg>=80) printf ("grade A");
else If (avg>=60&&avg<80) printf("Grade B");
else if (avg>=50&&avg<60) printf("Grade C");
else if (avg>=40&&avg<50) printf("Grade D");
else if (avg<40) printf("Grade E");
```

- Write a program to input employee name, hours worked and rate per hour then calculate gross pay = hours worked * rate per hour. Tax is charged based on gross pay as follows:

Gross pay	tax
Over 100000	20%
Between 50000 and 100000	10%
Below 50000	0

- Design a flow chart and a program to input employee name, rate per hour and hours worked then calculate basic pay, tax and net salary. Tax is charged as shown below:

Basic	tax
Over 100000	20%
Between 50000 and 100000	10%
Below 50000	0

Nesting *if …. Else* Statements

- When series of decisions are involved, we may have to use more than one ***if .. else*** statements in nested form. The general form for nested ***if…. else*** statement is;

```
if (test-conditions-1)
  { if (test-conditions-2);
        statement-1;
          else  statement - 2;  } statement-x ;
```

- If the condition-1 is true, condition-2 is tested and if condition-2 is true then and statement-1 will be evaluated otherwise statement-2 will be executed if condition-2 is false.
- If condition-1 is false control is transferred to the statement-x.

```
• If (gender==“female)
      {if (mean>=60) printf (“Admit student”);
        else printf (“don’t admit”);}
  Else printf (“Not admissible”);
```

- A bank‟s policy is to a 2% bonus on the ending balance at the end of the year (31st December) irrespective of the balance, and a 5% bonus is also given to female account holders if the balance is more than 50,000.

- main()
- if (sex is female)
- { if (balance > 50000)
- { bonus = 0.05 * balance; else bonus = 0.02 * balance; } else { bonus = 0.02 * balance; } balance = balance + bonus;

The Switch statement

The switch statement selects from a number of alternatives. The switch statement has the following components.

- The switch statement- It includes the variable to be tested in brackets.
- Braces { }- Used to indicate the start and end of the switch statement block.
- Case statement- Each case statement specifies a value to compare with the value specifies in the switch statement.
- Default – It is an optional statement included to indicate the message or statement to be executed if none of the case statements were matched.

Syntax

```
switch (value){
case 1: first alternative statement;
break;
case 2: second alternative statement;
break;
case 3: last alternative statement;
break;
default: default statement;
}
```

- NB: Switch statement can only be used for ordinal data types such as character and integer data types. Float data types are not allowed as switch values.

Example

1. Write a Program to input a number then display the number in words. (1 to 5).

2. Write a program to input grade. If grade is A, it should display Excellent, grade B - Good, Grade C – Average and D- Below average.

```
# include <stdio.h>
int main()
{ int num;
Printf(“Enter number”);
Scanf(“%d”,&num);
Switch (num)
{ case 1: printf(“One”);
  break;
  case 2: printf(“Two”);
  break;
case 3: printf(“Three”);
break;
case 4: printf(“Four”);
break;
case 5: printf(“Five”);
break;
default: printf(“Error”);
}
}
```

```
# include <stdio.h>
int main()
{ int num;
Printf(“Enter number”);
Scanf(“%d”,&num);
If (num==1)
printf(“One”);
If (num==2)
printf(“Two”);
If (num==3)
printf(“Three”);
If (num==4)
printf(“Four”);
If (num==5)
printf(“Five”);
}
```

- Write a program to input two numbers and a sign. Then perform calculations according to the sign entered as shown below.

Sign	Calculation
+	Add the two number
-	Subtract the two numbers
*	Multiply the two numbers
/	Divide the two numbers

```
# include <stdio.h>
int main()
{ int x,y;
   float ans;
   char sign;
Printf(“Enter number”);
Scanf(“%d”,&x);
Printf(“Enter 2nd number”);
Scanf(“%d”,&y);
Printf(“Enter sign”);
Scanf(“%c”,&sign);
Switch (sign)
{ case ‘+’: ans=x+y;
  break;
case ‘-’: ans=x-y;
  break;
case ‘*’: ans=x*y;
  break;
case ‘/’: ans=x/y;
  break;
  default: printf(“Error”);
Printf ( “answer %f”,ans);
  }
  }
```

Iteration control structure

- The purpose of loop statements is to *repeat statements* a given number of times until certain conditions occur.
- A programming loop is one that forces the program to go back up again and thus you can execute lines of code repeatedly.

There are three kinds of loop statements in C.
That is:

- while
- do..while
- for

while loop

The while statement continually executes a block of statements while a particular expression/condition is true. Its

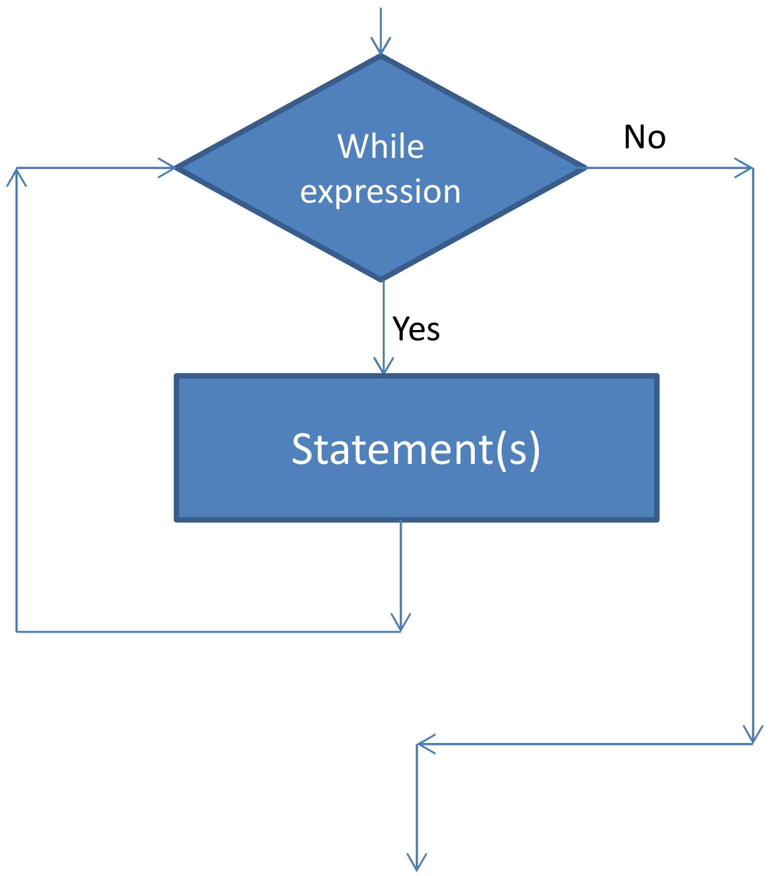
syntax can be expressed as:

```
while (expression)
{
statement(s)
}
```

NB: In while loop, the test is done at the start.i.e. test then execute.

- The while statement evaluates *expression*, which must return a boolean value. If the expression evaluates to true, the while statement executes the *statement*(s) in the while block.
- The while statement continues testing the expression and executing its block until the expression evaluates to false.

while flow chart



counter

- A counter is an integer variable that counts the number of loops.
- It is initialized before the loop and incremented or decremented within the loop.
- It is used when dealing with predefined number of loops.

Examples

- Write a program to display the word “hallo” 10 times.
- Write a program to display the numbers 1,2,3,4,5,6,7,8.
- Write a program to display the numbers 2,4,6,8………12.
- Write a program to input student name, maths, english and kiswahili and calculate total and average marks of five students.

```
# include <stdio.h>
Int main()
{ int counter;
    counter = 0;
   while(counter<=10)
           { printf (“ \nHallo”);
            counter++;
            }
}
```

```
# include <stdio.h>
Int main()
{ int counter;
    counter = 0;
   while(counter<=12)
          { printf (“%d”, counter);
           counter=counter +2;
          }
}
```

```
# include <stdio.h>
Int main()
{ int counter;
    counter = 10;
   while(counter>=1)
          { printf (“%d”,counter);
             printf (“\n”);
            counter--;
           }
}
```

```
#include <stdio.h>
int main()
{int count;
  int mat,eng,kis,tot; float avg;
 char sname[10];
 count=0;
  while (count<5)
   { printf (“Enter student
name”);
      scanf(“%s”,&sname);
     printf (“Enter
mathematics”);
      scanf(“%d”,&mat);
     printf (“Enter English”);
      scanf(“%d”,&eng);
      printf (“Enter Kiswahili”);
       scanf(“%d”,&kis);
       tot=mat+eng+kis;
       avg=tot/3;
 Printf (“Total is %d \n”,tot);
Printf (“Average is %f \n”,avg);
Count++
}
}
```

```
# include <stdio.h>
Int main()
{ int counter;
   counter = 10;
         do { printf (“%d”,counter);
            printf (“\n”);
           counter=counter-1;
          } while(counter>=1);

}
```

do while

C programming language also provides a do-while statement, which can be expressed as follows:

```
do
 {
 statement(s)
  } while (expression);
```

- The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top.
- Therefore, the statements within the do block are always executed at least once.

```
int main{
char reply; int prc,qty,tprc; char pname[8];
do{ printf( “Enter product name”);
       scanf (“%s”,&pname);
       printf( “Enter price”);
       scanf (“%d”,&prc);
        printf( “Enter Quantity”);
       scanf (“%d”,&qty);
        tprc=prc*qty;
        printf(“Total price %d \n”,tprc);
        printf(“ Do want to continue [Y/N]”);
        scanf (“%c”,reply);
     } while (reply==‘y’||’Y’);
}
```

for Loop

- A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
- A for loop is useful when you know how many times a task is to be repeated.

Syntax:

The syntax of a for loop is:

```
for(initialization; Boolean_expression; increment)
{
  Statements
}
```

Example

Write a program to display numbers 10,11,12,13,14……..19.

```
int main()
{
for(int x = 10; x < 20; x++)
{
printf("value of x : %d " , x );
printf("\n");
}
```

- Write a program to display numbers 10,9,8,7…1.
- Write a program to enter employee name, hours worked and rate per hour then calculate gross salary of 4 employees.