Chapter 5

Control Statements

In most of the C programs we have encountered so far, the instructions that were executed in the same order in which they appeared within the program (Sequentially). Each instruction was executed one and only once. Program of this type are unrealistically simple, since they do not include any logical control structures. But in realistic C programs, the program needs logical condition to determine if certain conditions are true or false, they do require the repeated execution of groups of statements and, they involve the execution of individual groups of statements on a selective basis. And all these operations can be carried out using the various control statements included in C. The different types of control statements are:

a. Branching or Selection Statement
b. Iteration or Looping Statement
c. Jump Statement

a. Branching or Selection Statement

In realistic C program, it may require a logical test to be carried out at some particular point within the program. Then depending on the outcome of the logical test, several possible actions will then be carried out, this is known as branching.

There is a special kind of branching, called selection in which one group of statements is selected from several available groups. C supports two types of selection statements:

1. If Statement
2. Switch Statement

1. The if Statement

"if statement" is very useful and easy to use for decision making. It allows a particular statement or more than one statements (Block Statement) to be executed only when certain condition is fulfilled.

The *general form* of if statement is

```
if (expression){
   Statement;
}else{
   Statement;
}
```

Where, a statement may consist of a single statement, a block of statements, or nothing (in case of empty statements). The else clause is optional.

In the above general form if the expression is TRUE then the statement below if is executed and control jumps to the next statements following the if statement. Thus, the statement following the else is ignored. If the expression is FALSE then the statement below if is ignored and the statement following the else is executed.

Example#1: Simple if statement

```
if (logical expression){
   S1;
   S2;
   .
   .
   Sn;
}
```

Example#2: if else statement

```
if (logical expression){
   S1;
   S2;
   .
   .
   Sn;
}else{
   S1;
   S2;
   .
   .
   Sm;
}
```

Example#3: Nested if else statement

```
if (logical expression){
   if (logical expression){
     Statement;
   }else{
     Statement;
   }
}else{
          Statement;
}
```

Example#4: Picking largest of three numbers

```
main(){
  int a, b, c;
  printf(" Enter the value of a, b, and c ");
  scanf(" %d %d %d ", &a, &b, &c);
  printf(" a = %4d, b = %4d, c = %4d ", a, b, c);
  if (a>b){
    if (a>c)
        printf(" a = %4d \n", a);
    else
        printf(" c = %4d \n", c);
    }
  else{
    if (b>c)
        printf(" b = %4d \n", b);
    else
         printf(" c = %4d \n", c);
    }
  getch();
  }
```

Example#5: Alternative method to pick the largest of three numbers

```
#include<stdio.h>
#include<conio.h>
void main(){
  int a, b, c, big;
  printf(“ enter the value of a, b, and c”);
  scanf(“ %d %d %d “, &a, &b, &c);
  big = a;
  if (b>big)
        big = b;
  if (c>big)
      big = c;
  printf(“ \n the largest number is %4d “, big); getch();
}
```

Example#6: Largest of four numbers using nested if-else

```
#include<stdio.h>
#include<conio.h>
void main(){
 int a,b,c,d;
 printf("Enter four numbers:");
 scanf("%d%d%d%d",&a,&b,&c,&d);
 if(a>b){
            if(a>c){
                        if(a>d)
                                    printf("Maximum is %d", a);
                        else
                                    printf("Maximum is %d",d);
            }else{
                  if(c>d)
                      printf("Maximum is %d",c);
                  else
                      printf(“Maximum is %d”,d);
```

```
      }
          }
 else{
    if(b>c)
          {
           if(b>d)
         printf("Maximum is %d",b);
          else
              printf("Maximum is %d",d);
           }
   else
       if(c>d)
           printf("Maximum is %d",c);
else
          printf("Maximum is %d",d);
}
getch();
 }
```

Example#7: Largest of four numbers using if-else and && operator

```
#include<stdio.h>
#include<conio.h>
void main(){
 int a,b,c,d;
 scanf("%d%d%d%d",&a,&b,&c,&d);
 if(a>b && a>c && a>d)
           printf("maximum is%d",a);
 else if(b>a && b>c && b>d)
           printf("maximum is %d",b);
 else if(c>a && c>b && c>d)
           printf("Maximum is %d",c);
 else
           printf("Maximum is %d",d);
  getch();
}
```

Example#8: Program to calculate the number of days in the months.

- " 30 days has September, April, June, and November"
- " All the rest have 31 days except February alone"
- " February have 29 days in Leap year and 28 in other years"

The program will ask the user to enter month and year and the rest of the calculations are performed using if-else condition.

```
#include<stdio.h>
main(){
  int month, years, days;
  clrscr();
  printf(" Enter the month: ");
  scanf(" %d ", &month);
  printf(" \n Enter the years: ");
  scanf(" %d ", &years);
  clrscr();
  if (month == 2){
    if ( (years%400 == 0) ||(year%4==0 && year%100!=0))
      days = 29;
    else
      days = 28;
    }
  if ( month != 2){
    if ((month == 4) || (month == 6) || (month == 9) || (month == 11))
      days = 30;
    else
      days = 31;
    }
printf(" number of days in the year %d and month %d is : %d", year, month, days);
getch();
}
```

The ? Alternative - You can use the ? operator to replace if-else statements of the general form:

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

However, the target of both if ans else must be a single statement not group of statement. The **?** operator is also called ternary operator because it requires three operands. It takes the general form:

Exp1 ? Exp2 : Exp3

where, Exp1, Exp2, Exp3 are expressions.

Here, the Exp1 is evaluated. If it is TRUE, Exp2 is evaluated and become the value of the entire ? expression. If Exp1 is FALSE, then Exp3 is evaluated and its value becomes the value of the expression.

For Example,

Consider,

```
X = 10;
Y = X>9 ? 100 : 200;
```

Output:

```
Y = 100
```

Same code written with the if-else statement would be

```
X = 10;
if  (X>9)
  Y = 100;
else
  Y = 200;
```

Example#9: Find the square of the given number

```
#include<stdio.h>
void main( ){
  int sqrd, n;
  printf(“ enter a number: “);
  scanf(“ %d “, &n);
  sqrd = n>0 ? n*n : - (n*n);
  printf(“ %d squared is %d “, n, sqrd);
}
```

2. The switch Statement

C has a built-in multiple - bench selection statement called switch . It is useful when on out of set alternative action is to be taken based on the value of an expression . It is particularly useful when variable values are classified with codes . The switch statement cause to be chosen from several available groups. The selection is based upon the current value of an expression that is included within the switch statement. The switch statement evaluates an expression & then transfer control to the following statement whose expression equals the evaluated expression. The general form of switch statement is given bellow :

```
switch(expression){
    case constant_1:
            statement1;
            statement2;
            break;
    case constant_2:
            statement1;
            break;
    case constant_3:
            statement1;
            break;
    default:
            statement;
}
```

Example#10: Program displaying day using the switch statement depending upon the number entered.

```
#include <stdio.h>
#include <conio.h>
main(){
  int choice;
  printf(“Enter the number of day:”);
  scanf(“%d”,&choice);
  switch(choice){
          case 1:  printf("The day you've chosen is Sunday");
                   break;
          case 2:  printf("The day you've chosen is Monday");
                   break;
          case 3:  printf("The day you've chosen is Tuesday");
                   break;
          case 4:  printf("The day you've chosen is Wednesday");
                   break;
          case 5:  printf("The day you've chosen is Thursday");
                   break;
          case 6:  printf("The day you've chosen is Friday");
                   break;
          case 7:  printf("The day you've chosen is Saturday");
                   break;
          default:  printf("Invalid option given");
  }
  getch();
}
```

Iteration or Looping Statement

In C, the program may require that a group of statements be executed repeatedly, until some logical condition has been satisfied this is known as looping or iteration. Some times the required number of repetition is known in advance; and sometimes the computation indefinitely until the logical condition becomes true.

There are three methods for generating repetition of a certain part of the program

a) for loop
b) while loop
c) do…while loop

a) for loop

The for loop is the most commonly used statement in C++. This loop consists of three expression. The first expression issued to initialize the index value, the second to check whether the loop is to be continued again and third to change the index value for further repetition.

```
for( count = 0 ; count < 5 ; count++ )
```

Example#1: *A program to display the numbers from 0 to 10 using for loop*

```
#include <stdio.h>
#include<conio.h>
void main(){
   int i=0;
   for(i=0;i<=10;i++)
   printf("%d",i);
   getch();
}
```

Example#2: *A program to find the sum and the average of given numbers.*

```
#include<stdio.h>
#include<conio.h>
void main(){
   int n,i;
   float sum=0.0,a,average;
   printf("How many numbers you want?");
   scanf("%d",&n);
   for(i=0;i<=n-1;i++){
```

```
    printf("Enter a number:");
    scanf("%f",&a);
    sum+=a;
    }
   average=sum/n;
   printf("Sum=%f",sum);
   printf("\nAverage= %f",average);
   getch();
}
```

Example#3: *A program to generate number in pyramid format*

```
#include<stdio.h>
#include<conio.h>
void main(){
  int p,q,a,k,l;
  printf("\nEnter the value for pyramid:");
  scanf("%d",&a);
  for(p=1;p<=a;p++)    /*for a row implement*/
  {
    for(q=a;q>p;q--)     /*for the space(s)*/
      printf(" ");
    for(k=1;k<=p;k++)        /*for previously printed number*/
      printf("%d",k);
    for(l=p-1;l>=1;l--)      /*for printing new nos.*/
      printf("%d",l);
    printf("\n");
  }
getch();
}
```

b. while loop statement

The second type of loop statement is while loop. While loop first checks whether the initial condition is true r false and finding it to be true, it will enter the loop and execute the statement

For single statement

```
while(condition)
    statement;
```

For compound statement

```
while(condition){
    statement1;
    statement2;
}
```

The while loop doesn't have initialization and increment part like for loop has.

Example#4: *Program to print numbers from 1 to 10*

```
#include<stdio.h>
#include<conio.h>
main(){
   int number=1;
   while(number<=10){
      printf("%d",number);
      number++;
   }
   getch();
}
```

Example#5: *A Program to find factorial of a given number*

```
#include<stdio.h>
#include<conio.h>
main(){
int a=1,n;
long float fact=1;
printf("Enter the value for n:");
scanf("%d",&n);
while(a<n){
```

```
a++;
fact*=a;
}
printf("Factorial of given no. is: %ld",fact);
getch();
}
```

c. do…while loop

do...while loop is another repetitive loop used in C and C++ programs. In case of do..while loop, as it enters loop at least once and then checks whether the give condition is true or false. As long as the test condition is true, statements will be repeated again and again, otherwise loop will terminate.

The general syntax of do...while loop is

```
do{
    statement1;
    statement2;
}while(condition);
```

Example#6: *A program to find the sum of odd numbers using do..while loop*

```
#include <stdio.h>
#include<conio.h>
void main(){
   int n,sum=0,i=1;
   printf("Enter the value of n:");
   scanf("%d",&n);
   do{
      sum+=i;
      i+=2;
   }while(i<=n);
   printf("The sum of odd number is %d",sum);
   getch();
}
```

Example#7: *A program to accepts a character and returns its ASCII value*

```
#include <stdio.h>
#include <conio.h>
main(){
   char somechar,answer;
   int asc_value;
   do{
      printf("Enter any character:");
     scanf(“%c”, &somechar);
      asc_value=somechar; /*assigning a character to an integer stores its ASCII value*/
      printf(“\nThe ASCII value of alphabet:",asc_value);
      printf("\nDo you want to input another character?");
      scanf(“%c”,&answer);
   }while(answer=='y' || answer=='Y');
  getch( );
}
```

Jump Statement

C has four statements that perform an unconditional branch. They are:

i. Break Statement
ii. Continue Statement
iii. Goto Statement
iv. Return Statement

i. The break Statement

In C programming, we often come across situations where we want to jump out of a loop instantly, without going back to the conditional test. In such situation keyword break allows us to do this. When the keyword break is encountered inside any C loop, control automatically passes to the first statement after the loop. In other words, the break statement is used to terminate loops or to exit from a switch. It can be used within a for, while, do-while, or switch statement.

The break statement is written simply as: **break;** without any embedded expressions or statements.

Example#1: *Program to determine whether a number is prime or not.*

A prime number is one which is divisible only by 1 and itself. To find prime number we have to divide a given number by all numbers from 2 to one less that itself. If remainder of any of these divisions is zero, the number is not prime.

```
#include<stdio.h>
#include<conio.h>
void main(){
   int num, i,flag=0;
   printf(" Enter a number: ");
   scanf("%d", &num);
   for(i=2;i<=num/2;i++){
     if(num% i == 0){
                  flag=1; break;
                 }
     }
   if (flag==0){    printf("\n prime number");
   }else{             printf("\n not prime number");
   }
  getch();
  }
```

The keyword *break* breaks the control only from the loop in which it is places. Consider the following program:

Example#2:

```
#include<stdio.h>
void main(){
        int i=1, j=1;

  while(i++<=100){
    while(j++<=200){
    if (j == 150){
          break;
    }else{
    printf("%d %d \n", i, j);
   }
  }
  }
getch();
}
```

In this program when j equals to 150, break takes the control outside the inner while only, since it is places inside the inner while.

ii. The continue Statement

In C programming, sometimes we want to take the control to the beginning of the loop, by passing the statement inside the loop which has not yet been executed. The keyword continue allows us to do this.

When the keyword continue is encountered inside any C loop, control automatically passes to the beginning of the loop. The continue statement can be included within a while, do-while or a for statement. It is written simply as: **continue;** without any embedded statements or expression.

Example#3: *Calculate the average of the nonnegative numbers in a list of numbers*

```
#include<stdio.h>
#include<conio.h>
void main(){
   int n,i,count=0;
   float x, average, sum=0;

  /* Initialize and read in a value for n */
   printf(" How many numbers?");
   scanf("%d", &n);
  /* Read in the values of x */
  for(i=1;i<=n;i++){
        printf("X = ");
        scanf(" %f ", &x);
      if(x < 0)
          continue;
       sum += x;
       ++count;
    }
   /* calculate the average and write out the answer */
   average = sum/count;
   printf("\n the average is %f \n", average);
   getch();
   }
```

Output:

```
How many numbers?  6
X = 1
X = -1
X = 2
X = -2
X = 3
X = -3
```

Note that the average would have been zero if all of the numbers had been averaged. But in this case only the positive numbers are summed and count is incremented so, the output is 2.000000.

iii. The goto Statement

In a difficult programming situation it seems so easy to use a goto to take the control where you want to. However, almost always, there is a more elegant way of writing the same program using if, for, while, and switch. These constructs are far more logical and easy to understand. So avoid using goto statements! They make a C programs life miserable.

A goto statement can cause program control to end up almost anywhere in the program, for reasons that are often hard to unravel. With good programming skills, goto can always be avoided.

The general form of goto statement is

goto label;

…

label:

where label is any valid label either before or after goto.

Example#4: *Check whether a person can vote or not*

```
#include<stdio.h>
#include<conio.h>
void main(){
  int  age;
  printf("Enter age: ");
   scanf(" %d ", &age);
    if (age<18)
        goto  input;
    else{
            printf("\n you can vote ");
            exit(0);
    }
    input: printf(" \n You cannot vote ");
    getch();
}
```

iv. The return Statement

The return statement is used to return from a function. It is categorized as a jump statement became it cause execution to return (jump back) to the point at which the call to the function was made. A return may or may not have a value associated with it. If return has a value associated with it, that value becomes the return value of the function.

In C, a non-void function does not technically have to return a value. If no return value is specified, a garbage value is returned. In C, if a function is declared as returning a value, it is good practice to actually return one. The general form of the return statement is

return expression;

The expression is present only if the function is declared as retuning a value. In this case, the value of expression will become the return value of the function. A function expression declared as void may not contain a return statement that specifies a value. Since a void function statement has no return value, it makes sense that no return statement within a void function can return a value.

Example#5: Program to check whether the String is Palindrome or not

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main(){
  char name[50];
  int i,l,flag=1;
  puts("Enter String:");
  gets(name);
  l=strlen(name);
  for(i=0;i<(l/2);i++){
    if(name[i]!=name[l-1-i]){
              flag=0; break;
            }
  }
  if(flag==1){    printf("The String is Palindrome");
  }else{              printf("The String is not a palindrome"); }
  getch();
}
```