Control Statements

Dr. S. R. Potdar
A. S. C. College, Rahata
potdarsatyajit@gmail.com
Introduction

In previous chapters we have studied basics of C language.

The C program is a sequence of various statements.

These statements can be divided into three categories:

1) Expression statements (followed by semicolon ;)
2) Compound statement which consists of various other statements enclosed in braces { }.
3) Control statements which decides order of statements to be executed.
Control statements

The control statements in c program are useful in the situation of decision making, loop control and program control.

There are three types of control, statements:

1) Loop Control statements:
   while, do....while, for.

1) Decision Control statements:
   if, if...else, switch.

1) Jump control statements:
   break, continue, exit, goto.
Loop control Statements

A segment of program code that is executed repeatedly is called as a Loop.

The repetition is done until some condition is satisfied.

There are two types of loops:
1) Top tested/entry controlled loop- Condition is tested before loop execution.
2) Bottom tested/exit controlled loop- Condition is tested after loop execution.

There are three loop statements/structures in C:
1) while statement,
2) do.....while statement,
3) for statement.
while statement

It is simplest looping structure and called while loop. It is used when one don’t know number of times the loop to be executed, but depends upon some test condition.

The while is also called entry controlled loop, since the test condition is executed first. If test condition is true, then body of loop is executed. If test condition false, the control is passed out of loop.

The general format of while statement is:

```java
while (test condition) {
    body of loop;
}
```
Flow chart of while loop

Start

Initialization

Is condition True?

YES

Statement

NO

Stop
Example

Write a Program to display digits 0 through 5 using while loop.

```c
#include<stdio.h>
#include<conio.h>

void main()
{
    int digit=0;
    while(digit<=5)
    {
        printf("%d\n", digit);
        ++digit;
    }
    getch();
}
```

Output of Program:

```
0
1
2
3
4
5
```
```c
#include<stdio.h>
#include<conio.h>

void main()
{
    int digit=0;
    while(digit<=5)
    {
        printf("%d\n", digit++);
        getch();
    }
}
```

Output of Program:

0
1
2
3
4
5
**do while loop**

The **do while loop** is an *exit controlled loop*, that is, it **first executes** body of loop and **then test** the condition. If condition is **true** again **loop** is executed. If condition **false**, control comes **out of loop**.

The general form of do while is:

```java
do
{
    body of loop
}
while(condition);
```
Start

Initialization

Statement

Is condition True?

YES

NO

Stop
Example

Write a Program to display digits 0 through 5 using do while loop.

```c
#include<stdio.h>
#include<conio.h>

void main()
{
    int digit=0;
    do
    {
        printf("%d\n", digit++);
    } while(digit<=5);
    getch();
}
```

Output of Program:

0
1
2
3
4
5
for statement

It is **flexible, powerful** and hence most commonly used loop structure in C.

The general form of for loop is:

```c
for(exp1;exp2;exp3)
```

Where `exp1` - is used to initialize a parameter which controls the loop. *(assignment statement)*

`exp2` - is a condition that must be true for looping. *(logical statement)*

`exp3` - is used to alter parameter or its value in `exp1`. *(unary or assignment statement)*
General format of for loop

```c
for(e1; e2; e3)
{
    statement;
}
```

Comparison with while

```c
e1;
while(e2)
{
    {
        statement;
    }
    e3;
}
```
Initialization Expression 1

Is condition True? (Exp2)

YES

Statement

Expression 3

NO

Stop
Example

Write a program to display digits 0 through 5 using for loop.

```
#include<stdio.h>
#include<conio.h>

void main()
{
    int digit;
    for(digit=0; digit<=5; digit++)
        printf("%d\n", digit);
    getch();
}
```

Output of Program:
0
1
2
3
4
5
Nesting of loop

Nesting of loop is a structure in program, in which one for statement is present within another for statement.

Structure:
Example-
/* WELCOME */
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j;
    for(i=1;i<=3;i++)
        for(j=1;j<=2;j++)
            printf("WECOME\n");
    getch();
}

This program will display WELCOME six times on screen.
Decision making Statements

These statements are used where one has to **test a condition** and then selects an **alternative path**.

There are **three types**

i) **if** statement

ii) **if...else** statement

iii) **switch** statement
Simple if statement

It is the simplest decision making statement. The general form is

```java
if(test_condition)
{
    statement_true;
}
statement_x;
```

Thus if statement evaluates the condition,
If it is true, then `statement_true` is executed first and then `statement_x` is executed,
But if condition false then it executes `statement_x` only.
Flow chart

Entry

Test Condition?

True

Statement_true

False

Statement_x

Next
Example-
/* if statement */
#include<stdio.h>
#include<conio.h>
void main()
{
    int x;
    printf("\n Enter a number=");
    scanf("\n %d", &x);
    if(x>5)
    {
        printf("\n The number is greater than 5");
    }
    getch();
}
The if.....else statement

It is useful in decision making when there are two alternatives in program that is one for true and other when condition false.

General form:

```python
if(test_expression)
{
    Statement_true;
}
else
{
    statement_false;
}
statement_x;
```
Flow chart

Entry

Test Condition?

True

Statement_true

Statement_x

Next

False

Statement_false
Example-

/* if….else statement */
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    printf("Enter two numbers=\n");
    scanf("%d %d", &a, &b);
    if(a<b)
    {
        printf("a is less than b\n");
    }
    else
    {
        printf("a is greater than b\n");
    }
    getch();
}
Nesting of if....else

When a series of decisions are involved, one can use more than one if....else statements in nested form. It provides flexibility in programming.

There are various forms of nesting as follows:

1)

```
if(exp1)
{
    statement_1;
}
else
{
    if(exp2)
    {
        statement_2;
    }
}
```
2) if(exp 1)
    {
        if(exp2)
            statement_1;
        else
            statement_2;
    }
else
    statement_3

3) if()
    {
        if()
            statement1;
        else
            statement2;
    }
else
    statement_3

    {
        if()
            statement3;
        else
            statement4;
    }
Flow chart of Nested if …else statement

Entry

Test cond 1

True

Test cond 2

False

Statement 3

Statement 2

Statement 1

Statement X

Next
/* Nesting if...else statement */
/* Largest of Three numbers*/

#include<stdio.h>
#include<conio.h>
#include<math.h>

void main()
{
    int a, b, c;
    clrscr();
    printf("Enter three numbers \n");
    scanf("%d %d %d", &a, &b, &c);
    printf("\n The Largest number is=");
    if(a>b)
    {
        if(a>c)
            printf("%d", a);
        else
            printf("%d", c);
    }
    else
    {
        if(b>c)
            printf("%d", b);
        else
            printf("%d", c);
    }
    getch();
}
If number of alternatives are increasing, it is difficult to take proper decision using if/ if..else.

In C we can use switch statement in such situation.

The switch statement test the value of given variable against a list of case value and when a match is found, the block of statement associated with that case is executed.
The general form of switch statement is

```
switch(exp)
{
    case value1:
    {
        block1
        break;
    }
    case value2:
    {
        block2
        break;
    }
    case valuen:
    {
        blockn
        break;
    }
    default:
    {
        block_default
        break;
    }
}
statementX;
```
/* switch statement*/

#include<stdio.h>
#include<conio.h>
#include<math.h>

void main()
{
    int a, b, c;
    clrscr();
    printf("Enter any two numbers=");
    scanf("%d %d", &a, &b,);
    printf("n 1:Addition");
    printf("n 2:Subtraction");
    printf("n 3:Multiplication");
    printf("n Enter your choice=");
    scanf("%d", &c);
    switch(c)
    {
    case 1:
    {
        printf("n The addition=%d", a+b);
        break;
    }
    case 2:
    {
        printf("n The subtraction=%d", a-b);
        break;
    }
    case 3:
    {
        printf("n The multiplication=%d", a*b);
        break;
    }
    default:
    {
        printf("n Your option is wrong.");
        break;
    }
    }
    getch();
}
While using switch statement following points are important:

1) We can put cases in any order
2) Character/integers are allowed in switch (no float).
3) Mixing of integer and char is allowed in cases.
4) Sometimes there may not be any statement in switch.
5) If no default case then program fails and executes next instruction.
6) <=, >= can't be used.
7) Break statement takes control of program out of switch. Continue statement can't go to beginning of switch.
8) Nesting is possible
9) Used in menu selection
Jump Control statements

These statements are useful to change program flow unconditionally.

When these statements are encountered in program, the program flow is discontinued and control of program execution is transferred to new set of statements directly.

There are four jump statements in C:
1) break
2) continue
3) exit()
4) goto
break statement

It is used to **terminate a loop** or to **exit from switch statement**.

When this statement is encountered the control of program is transferred to **first statement after the loop**.

It is used along with **switch** and within **for, while, do...while** statements.

**General form**

`break;`
1) 
```
while (cond1)
{
    ------------
    ------------
    if (cond2)
        break;
    ------------
    ------------
}
statementX;
```

2) 
```
do
{
    ------------
    ------------
    if (cond1)
        break;
    ------------
    ------------
}
while (cond2)
statementX;
```
3) 
for(e1;e2;e3) 
{ 
    ------
    ------
    if(cond)
    break;
} 
statementX;

4) 
for(e1;e2;e3) 
{ 
    ------
    ------
    for() 
    { 
        ----
        ----
        if(cond)
        break;
    } 
    statementX;
}
continue statement

It is used to **bypass** the remaining loop and control is transferred to beginning of program.

**general form**

```
continue;
```

1) ```
   while(cond1)
   {
      -----------------
      if(cond2)
      continue;
      -----------------
   }
   statementX;
```
2) do
   { if(cond2) continue; }
   while(cond2) statementX;

3) for(e1;e2;e3)
   { if(cond) continue; }
   statementX;
It is used to **change** the **control** of program unconditionally. The **jump** can **occur at any statement** within program without condition.

**general form**

```plaintext
goto LABEL;
LABEL:
```
1) Forward jump

```
goto label;
label:
```
exit statement

It causes immediate termination of entire program. It is used when error occurs in program. The general form is

- `exit();` or
- `exit(0);` or
- `exit(n);`

`n` is non-zero number that terminates the program due to some detected error. `exit(0)` or `exit()` terminates program normally.
the comma operator

It is used within for statement.

the general form is

\[ \text{for}(\text{exp1a}, \text{exp1b}; \text{exp2}; \text{exp3}) \]

Where exp1a and exp1b are two initialization expressions.
Library functions

These are functions in C program that carry out various operations or calculations.
Like
1) read/write, open/close etc.
2) conversion of upper/lower case characters
3) mathematical operations
4) string operations

These can be accessed by using header files like `stdio.h, conio.h, math.h, ctype.h,` etc.

Following table shows some commonly used library functions:
<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(i)</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>cos(d)</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>cosh(d)</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>exp(d)</td>
<td>double</td>
<td>e raise to d</td>
</tr>
<tr>
<td>getchar() / putchar()</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>log(d)</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>pow(d1,d2)</td>
<td>double</td>
<td>d1 raise to d2</td>
</tr>
<tr>
<td>printf/scanf</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>tolower(c) / toupper(c)</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>sqrt(d)</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>toascii</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>tan(d)</td>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>
Concluded....