Structured Programming using C

RCP2SFCES101

Unit-3

Control Structures

Contents

Decision making with Branching

Decision making with Branching

In C programming, following are the decision making and branching statements

- if statement
  1. Simple if statement
  2. if..else statement
  3. Nested if..else statement
  4. else if ladder
- switch statement

1. Simple if statement

- It is used to decide whether the certain block of statements will be executed or not.
- i.e. if certain condition is true then the block will be executed otherwise not.
- **Syntax:**

```
if (test expression)
{
    statement-block;
} statement-x;
```

- The statement-block may be single statement or a group of statements.
- If ***test expression*** is true, then the ***statement-block*** will be executed; otherwise the ***statement-block*** will be skipped and the execution will jump to the ***statement-x***.

Example: Simple if statement

```c
#include<stdio.h>
int main()
{
        int a, b, c, d;

        printf("Enter value of a: ");
        scanf("%d",&a);

        printf("Enter value of b: ");
        scanf("%d",&b);

        if(b > 0)
        {
            c=a/b;
            printf("\nC = %d",c);
        }
        d = a + b;
        printf("\nD = %d",d);

    return 0;
}
```

Output

G:\SPC\Practical Programs\simpleif.exe

```
Enter value of a: 10
Enter value of b: 5

C = 2
D = 15
--------------------------------
```

G:\SPC\Practical Programs\simpleif.exe

```
Enter value of a: 10
Enter value of b: 0

D = 10
--------------------------------
```

2. if..else statement

- The if...else statement is extension of simple if statement.

  Syntax:

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

- If ***test expression*** is true, then the ***statement-block-1*** immediately following if statement will be executed; otherwise, the ***statement-block-2*** will be executed.
- Either ***statement-block-1*** or ***statement-block-2*** will be executed, not both.
- In both cases, the control is transferred to the ***statement-x***.

Example: if..else statement

```c
//Program to imput a number and check whether it is odd or even

#include <stdio.h>
int main()
{
    int a;

    printf("Enter a number: ");
    scanf("%d", &a);

    // Check if the number is odd or even
    if (a % 2 == 0)
        printf("%d is an even number.\n", a);
    else
        printf("%d is an odd number.\n", a);

    return 0;
}
```

Output

```
G:\SPC\Practical Programs\Ex-2a EvenOdd.exe
Enter a number: 5
5 is an odd number.
--------------------------------
```

```
G:\SPC\Practical Programs\Ex-2a EvenOdd.exe
Enter a number: 8
8 is an even number.
--------------------------------
```

3. Nested if..else statement

When series of decisions are involved, then we have to use more than one if...else statement in nested form.

- **Syntax**:

```
if (test expression-1)
{
        if (test expression-2)
           statement-block-1;
        else
           statement-block-2;
}
else
{
        if (test expression-3)
           statement-block-3;
        else
           statement-block-4;
}
statement-x;
```

- If test expression-1 is true, then the test expression-2 will be evaluated and if the test expression-2 is true then statement-block-1 will be executed; otherwise, the statement-block-2 will be executed.
- If test expression-1 is false, then the test expression-3 will be evaluated and if the test expression-3 is true then statement-block-3 will be executed; otherwise, the statement-block-4 will be executed.
- Either one of the four statement-blocks will be executed.
- In all cases, the control is transferred to the statement-x.

Example: Nested if..else statement

```c
//Program to imput three numbers and find largest number

#include<stdio.h>
int main()
{
        int a, b, c;
        printf("Enter values of a, b and c: ");
        scanf("%d%d%d",&a,&b,&c);

        if(a > b)
        {
            if(a > c)
            printf("A is large");
            else
            printf("C is large");
        }
        else
        {
            if(b > c)
            printf("B is large");
            else
            printf("C is large");
        }
    return 0;
}
```

Output

```
G:\SPC\Practical Programs\LargestOfThree.exe
Enter values of a, b and c: 10 5 8
A is large
--------------------------------
```

```
G:\SPC\Practical Programs\LargestOfThree.exe
Enter values of a, b and c: 5 8 2
B is large
--------------------------------
```

```
G:\SPC\Practical Programs\LargestOfThree.exe
Enter values of a, b and c: 5 2 8
C is large
--------------------------------
```

4. else if ladder

- The else...if ladder is used when multipath decisions are involved.
- Multi-path decision is a chain of ifs in which the statement associated with each else is if statement.

- **Syntax**:

```
if (condition-1)
  statement-block-1;
else if (condition -2)
  statement-block-2;
  - - - - - - - - - - - -
  - - - - - - - - - - - -
else if (condition-n)
  statement-block-n;
else
  default-statement-block;

statement-x;
```

- The conditions are evaluated from top to downwards. As soon as true condition is found, the statement-block associated with that condition is executed and the control is transferred to statement-x.
- If all the n condition becomes false, then default-statement-block will be executed.

Example: else if ladder

```c
//Program to display class/grade of student from percentage.

#include<stdio.h>
int main()
{
    float per;

    printf("Enter percentage: ");
    scanf("%f",&per);

    if(per>=75)
      printf("Distinction");
    else if(per>=60)
      printf("First Class");
    else if(per>=40)
      printf("Second Class");
    else
      printf("Fail");
    return 0;
}
```

Output

```
G:\SPC\Practical Programs\GradeFromPer.exe
Enter percentage: 85
Distinction
--------------------------------
```

```
G:\SPC\Practical Programs\GradeFromPer.exe
Enter percentage: 62
First Class
--------------------------------
```

```
G:\SPC\Practical Programs\GradeFromPer.exe
Enter percentage: 42
Second Class
--------------------------------
```

```
G:\SPC\Practical Programs\GradeFromPer.exe
Enter percentage: 30
Fail
--------------------------------
```

switch statement

Syntax:

```
switch (expression)
{
    case  value-1:
                block-1;
                break;
    case  value-2:
                block-2;
                break;
    case  value-3:
                block-3;
                break;
    case  value-n:
                block-n;
                break;
    default:
                default-block;
}
statement-x;
```

- The ***expression*** is an integer expression or character.
- ***value-1, value-2,… value-n*** are constants are known as case labels. Each of these values should be unique within the switch statement
- ***block-1, block-2,….,block-n*** are statement blocks may contain zero or more statements. There is no need to put braces around these blocks but the case labels ends with colon(:)
- When switch is executed, the value of expression is successively compared against the values value-1, value-2, …, value-n.
- If a case is found whose value matches with the value of *expression*, then block of statements associated with that case are executed.
- The *break* statement at the end of each block indicates that the end of particular case and causes exit from switch statement, transferring the control to the statement-x.
- The *default* is optional case. When present, it will be executed if the value of expression does not match with any of the case values.

Example: switch statement (Integer expression)

```c
//Program to display pressed digit in words (Between 1 to 5)
#include<stdio.h>
int main()
{
    int digit;

    printf("Enter any digit: ");
    scanf("%d",&digit);

    switch(digit)
    {
        case 0: printf("Zero");
                break;
        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;
        case 6: printf("Six");
                break;
        case 7: printf("Seven");
                break;
        case 8: printf("Eight");
                break;
        case 9: printf("Nine");
                break;
        default:
                printf("Invalid digit");
    }
    return 0;
}
```

Output

```
G:\SPC\Practical Programs\SwitchCase.exe
Enter any digit: 5
Five
--------------------------------
```

Example: switch statement (Character expression)

```c
/* Program to display name of color 'RED' if user enters character 'r',
   'GREEN' if user enters character 'g' and 'BLUE' if user enters character 'g' */

#include<stdio.h>
int main()
{
    char color;
    printf("Enter color char (r, g or b): ");
    color = getchar();

    switch(color)
    {
        case 'r': printf("RED");
                  break;

        case 'g': printf("GREEN");
                  break;

        case 'b': printf("BLUE");
                  break;

        default:
                  printf("Invalid Color code");
    }
    return 0;
}
```

Output

```
G:\SPC\Practical Programs\SwitchCase1.exe
Enter color char (r, g or b): g
GREEN
--------------------------------
```

```
G:\SPC\Practical Programs\SwitchCase1.exe
Enter color char (r, g or b): p
Invalid Color code
--------------------------------
```

Looping Statements

Looping Statements

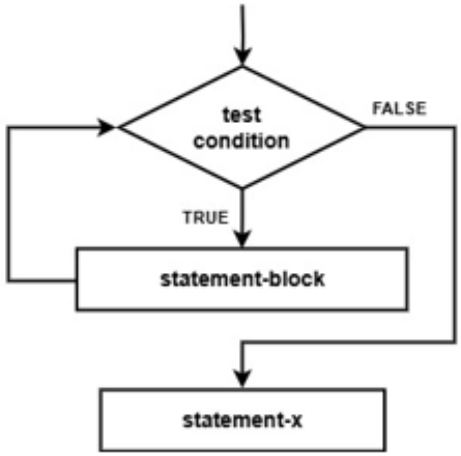
C programming supports following three looping statements:

1. while statement
2. do while statement
3. for statement

while statement

- The while is an entry-controlled loop statement.
- If the number of iterations are not fixed, then it is recommended to use while loop.
- The general form of while statement is:

```
Initialization;
while(test condition)
{
        //statement-block;
}
statement-x
```



- The test condition is evaluated and if test condition is true, then the statement-block is executed.
- The statement-block executed repeatedly until the test condition finally becomes false.
- The statement-block may have one or more statements.
- The braces { } are needed only if statement-block contains two or more statements.

Example: while statement

```c
// Program to display numbers from 1 to 10 using while loop

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

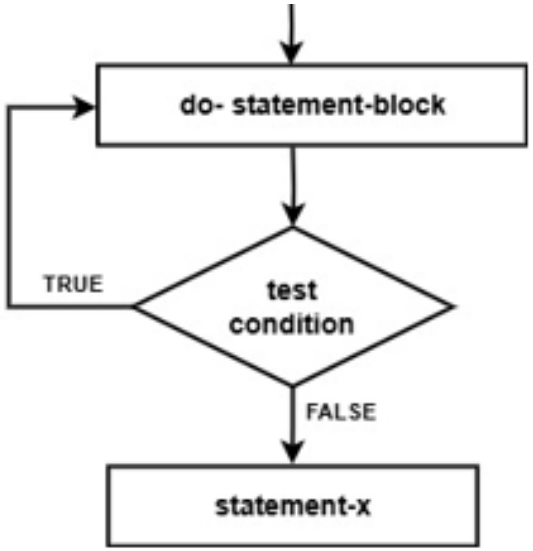
Output

```
G:\SPC\Practical Programs\while.exe
1
2
3
4
5
6
7
8
9
10
```

do..while statement

- The do-while is an exit-controlled loop statement.
- If the number of iterations are not fixed and you must have to execute loop at least once, then it is recommended to use do-while loop.
- The general form of do-while statement is:

```
Initialization;
do
{
    statement-block;
} while(test condition);
```



- The ***statement-block*** is executed first and then the ***test condition*** is evaluated.
- ***statement-block*** executed repeatedly until the ***test condition*** finally becomes false
- As do-while is exit-controlled loop statement, therefore the body of loop executed at least once.

Example: do..while statement

```c
// Program to display numbers from 1 to 10 using do while loop

#include<stdio.h>
int main()
{
    int n = 1;

    do
    {
        printf("%d\n",n);
        n++;
    }while(n <= 10);

    return 0;
}
```

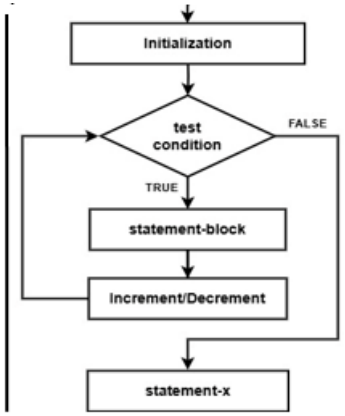
Output

```
G:\SPC\Practical Programs\dowhile.exe
1
2
3
4
5
6
7
8
9
10
```

for statement

- The for is an entry-controlled loop statement.
- If the number of iterations are fixed, then it is recommended to use for loop.
- The general form of for statement is:

```
for (initialization; test condition; increment/decrement)
{
  statement-block;
}
```



- The execution of for loop is as follows:
  1. Initialization of control variables is done first, using assignment statement such as i=1.
  2. Next the test condition is evaluated, and if the condition is true then the statement-block is executed.
  3. After executing statement-block, the control is transferred back to the for statement. Now the control variable is either incremented or decremented.
  4. Step 2 and 3 continues till test condition becomes false.

Example: for statement

```c
// Program to display numbers from 1 to 10 using for loop

#include<stdio.h>
int main()
{
    int n;

    for(n = 1; n<=10; n++)
    {
        printf("%d\n",n);
    }

    return 0;
}
```

Output

```
G:\SPC\Practical Programs\for.exe
1
2
3
4
5
6
7
8
9
10
```

Nested Control Structure

Nested Control Structure

- A nested loop refers to a loop that is contained within another loop.
- If the program has to repeat a loop more than once, then nested loop is used.
- In nested loops, the inner loop executes completely before the outer loop's next iteration.
- Each inner loop should be enclosed completely in the outer loop; overlapping loops are not allowed.
- You can define any type of loop inside another loop; for example, you can define while loop inside a for loop.

Example 1: Nested Control Structure

```c
/* Program to display following pattern.
            1
            1 2
            1 2 3
            1 2 3 4
            1 2 3 4 5       */

#include <stdio.h>
int main()
{
    int row, col;

    for (row=1; row<=5; row++)
    {
        for (col=1; col<=row; col++)
        {
            printf("%d ", col);
        }
        printf("\n");
    }
    return 0;
}
```

Output

```
G:\SPC\Practical Programs\pattern1.exe
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

--------------------------------
```

Example 2: Nested Control Structure

```c
/* Program to display following pattern.
        *
        * *
        * * *
        * * * *
        * * * * *                    */

#include <stdio.h>
int main()
{
    int row, col;

    for (row=1; row<=5; row++)
    {
        for (col=1; col<=row; col++)
        {
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

Output

```
G:\SPC\Practical Programs\pattern1.exe
*
* *
* * *
* * * *
* * * * *

--------------------------------
```

control statements

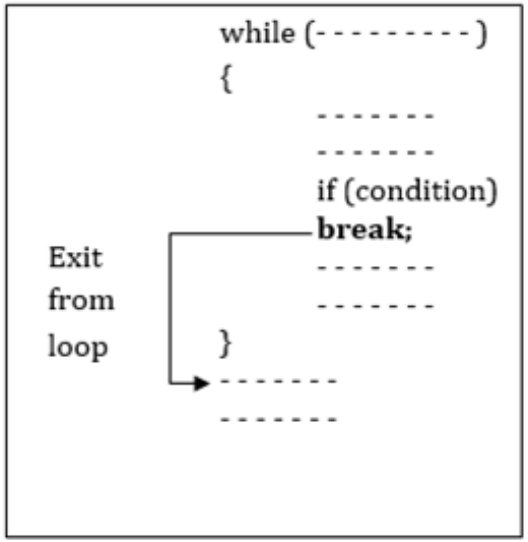
control statements

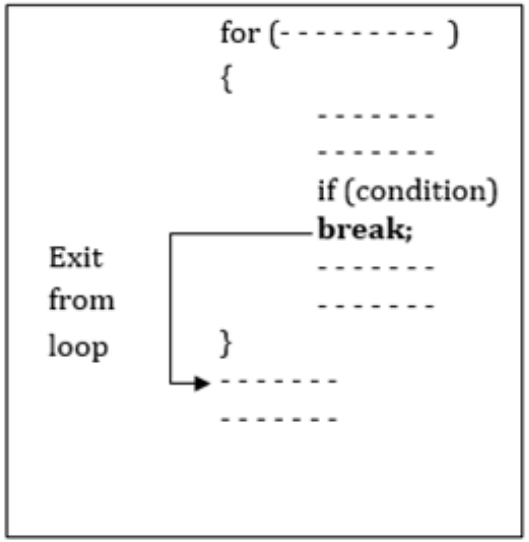
C programming supports following control statements:

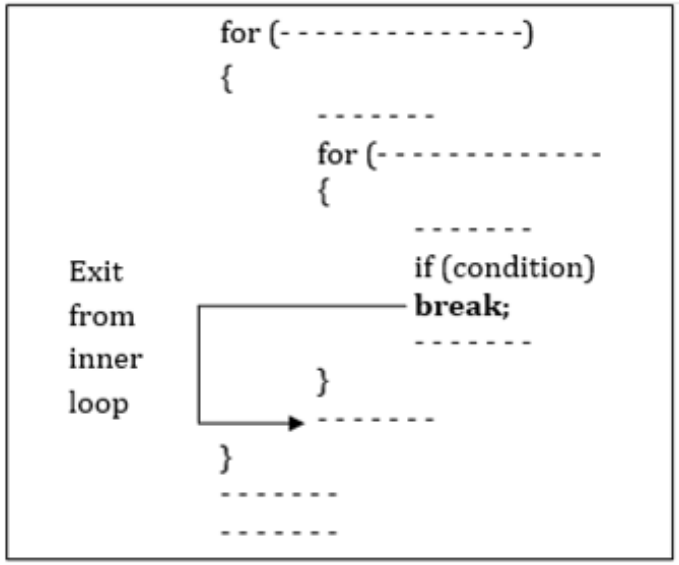
1. break
2. continue
3. goto

break statement

- The break statement is used to jump outside the current loop.
- When a break statement is encountered inside a loop, the loop is immediately terminated and the program continues with the statement immediately following the loop.
- When loops are nested, the break statement would only terminate the loop containing break statement.







Example: break statement

```c
// Program to demonstrate use of break statement

#include<stdio.h>
int main()
{
    int i;

    for(i=1; i<=10; i++)
    {
        if(i==5)
        break;

        printf("i = %d\n",i);
    }
    return 0;
}
```

Output

```
G:\SPC\Practical Programs\break.exe
i = 1
i = 2
i = 3
i = 4
```

continue statement

- The continue statement is used in loop when you need to jump to the next iteration of the loop immediately.
- The continue statement tells the compiler "Skip the following statements and continue with the next iteration".

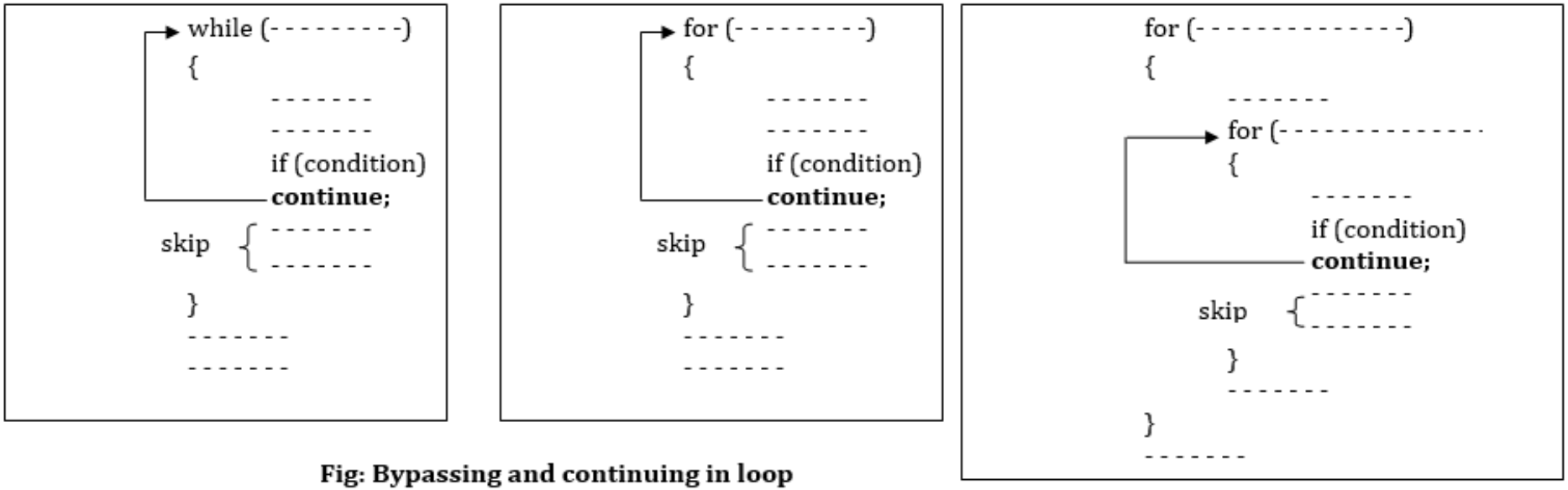


Fig: Bypassing and continuing in loop

Example: continue statement

```c
// Program to demonstrate use of continue statement

#include<stdio.h>
int main()
{
    int i;

    for(i=1; i<=10; i++)
    {
        if(i==5)
        continue;

        printf("i = %d\n",i);
    }
    return 0;
}
```

Output

```
G:\SPC\Practical Programs\continue.exe
i = 1
i = 2
i = 3
i = 4
i = 6
i = 7
i = 8
i = 9
i = 10
```

goto statement

- The goto statement is another type of control statement supported by C
- The control is unconditionally transferred to the statement associated with the label specified in the goto statement
- The form of a goto statement is

  goto ***label_name***;
- A statement label is defined in exactly the same way as a variable name
- The statement label must be followed by a colon (:)

Example: goto statement

```c
// Program to demonstrate use of goto statement

#include <stdio.h>
int main()
{
    int n;

    printf("Enter number: ");
    scanf("%d", &n);

    if (n < 0)
        goto negative;
    else
        goto positive;

positive:
    printf("The number is positive.\n");
    goto end;

negative:
    printf("The number is negative.\n");

end:
    printf("Program ended.\n");

    return 0;
}
```

Output

```
G:\SPC\Practical Programs\goto.exe
Enter number: 5
The number is positive.
Program ended.
--------------------------
```

```
G:\SPC\Practical Programs\goto.exe
Enter number: -4
The number is negative.
Program ended.
--------------------------
```

References

References

- "MASTERING C" by K.R.Venugopal and Sudeep R.Prasad , Tata McGraw-Hill Publications
- "Programming in ANSI C", by E. Balaguruswamy, Tata McGraw-Hill Education.
- "Let Us C", by Yashwant Kanetkar, BPB Publication.
- "Programming in C", by Pradeep Day and Manas Gosh, Oxford University Press.
- https://www.javatpoint.com/c-programming-language-tutorial
- https://www.tutorialspoint.com/cprogramming/index.htm
- https://www.geeksforgeeks.org/c-programming-language/
- https://www.youtube.com/playlist?list=PLdo5W4Nhv31a8UcMN9-35ghv8qyFWD9_S


Thank
you