Basic c programs

C Program to Print “Hello World”

```
// Header file for input output functions
#include <stdio.h>
// Main function: entry point for execution
int main()
{
   // Writing print statement to print hello world
   printf("Hello World");
   return 0;
}
```

Output:

```
Hello World
```

C Program To Print Your Own Name

```
// C Program to Print Your Own Name using printf
#include <stdio.h>
int main()
{
   // Printing your name "ROHIT" on the output screen
   printf("ROHIT");
   return 0;
}
```

Program to Print Your Own Name by Taking it as Input

```
// C Program to Print Your Own Name using scanf and printf
```

```
#include <stdio.h>
int main() {
// Defining string (character array) assuming 100
   // characters at max
   char name[100];
   // Taking input from the user
   printf("Enter Your Name: ");
   scanf("%s", name);
 // Printing your name to the screen
   printf("Your Name: %s\n", name);
 return 0;
}
```

C Program to Add Two Integers

```
// C program to add two numbers
#include <stdio.h>
int main() {
   int a, b, sum = 0;
      // Read two numbers from the user
   printf("Enter two integers: ");
   scanf("%d %d", &a, &b);
   // Calculate the addition of a and b
   // using '+' operator
   sum = a + b;
```

```
    printf("Sum: %d", sum);
    return 0;
}
```

Program to Print ASCII Value

```
#include <stdio.h>
int main() {
    char c;
    printf("Enter a character: ");
    scanf("%c", &c);

    // %d displays the integer value of a character
    // %c displays the actual character
    printf("ASCII value of %c = %d", c, c);

    return 0;
}
```

Swap Two Numbers

```
// C Program to Swap Two Numbers using a
// Temporary Variable
#include <stdio.h>
int main() {
    int a = 5, b = 10, temp;
    // Swapping values of a and  b
    temp = a;
    a = b;
    b = temp;
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

```
#include<stdio.h>
int main() {
  int first, second, temp;
  printf("Enter first number: ");
  scanf("%d", &first);
  printf("Enter second number: ");
  scanf("%d", &second);
  temp = first;
  first = second;
  second = temp;
  printf("\nAfter swapping, first number = %.d\n", first);
  printf("After swapping, second number = %.d", second);
  return 0;
}
```

CONDITIONAL BRANCHING STATEMENTS

The conditional branching statements help to jump from one part of the program to another depending on whether a particular condition is satisfied or not. These decision control statements include:

• if statement

• if-else statement

• if-else-if statement

• switch statement

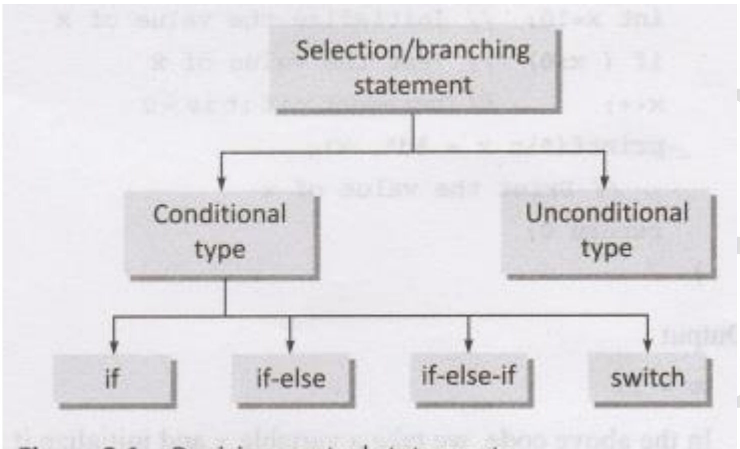
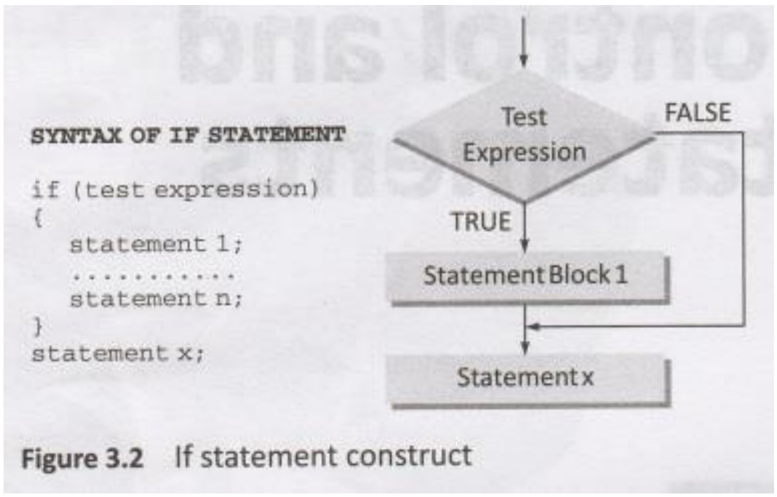


Figure 3.1 Decision control statements

If Statement

The *if* statement is the simplest form of decision control statements that is frequently used in decision making. The general form of a simple if statement is shown in Figure 3.2.

The *if* block may include one statement or n statements enclosed within curly brackets. First the test expression is evaluated. If the test expression is true, the statement of *if* block (statement 1 to *n*) are executed otherwise these statements will be skipped and the execution will jump to statement x.


SYNTAX OF IF STATEMENT

```
if (test expression)
{
   statement 1;
   ...........
   statement n;
}
statement x;
```

Figure 3.2 If statement construct

The statement in an *if* block is any valid C language statement and the test expression is any valid C language expression that may include logical operators. Note that there is no semicolon after the test expression. This is because the condition and statement should be put together as a single statement

Programming Tip: ***Properly indent the statements that are dependent on the previous statements.***

```
#include <stdio.h>
int main()
{
int x=10; // Initialize the value of x
if ( x>0) // Test the value of x
 x++; // Increment x if it is > 0
printf("\n x = %d", x); // Print the value of  x
return 0;
}
```

Output

x = 11

---

1. Write a program to determine whether a person is eligible to vote.

```
#include <stdio.h>
#include <conio.h>
int main()
{
int age;
printf("\n Enter the age: ");
 scanf("%d", &age);
if (age >= 18)
printf("\n You are eligible to vote");
getch();
return 0;
}
```

Output

```
Enter the age: 28
You are eligible to vote
```

If-Else Statement

We have studied that the *if* statement plays a vital role in conditional branching. Its usage is very simple, the test expression is evaluated, if the result is true, the statement(s) followed by the expression is executed else if the expression is false, the statement is skipped by the compiler.

But what if you want a separate set of statements to be executed if the expression returns a zero value? In such cases we use an *if-else* statement rather than using simple *if* statement. The general form of a simple *if-else* statement is shown in Figure 3.3.

Programming Tip: ***Align the matching if-else clauses vertically.***

In the syntax shown, we have written statement block. A statement block may include one or more statements. According to the *if-else* construct, first the test expression is evaluated. If the expression is true, statement block 1 is executed and statement block 2 is skipped. Otherwise, if the expression is false, statement block 2 is executed and statement block 1 is ignored. Now in any case after the statement block 1 or 2 gets executed the control will pass to statement x. Therefore, statement x is executed in every case.

SYNTAX OF IF-ELSE STATEMENT

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

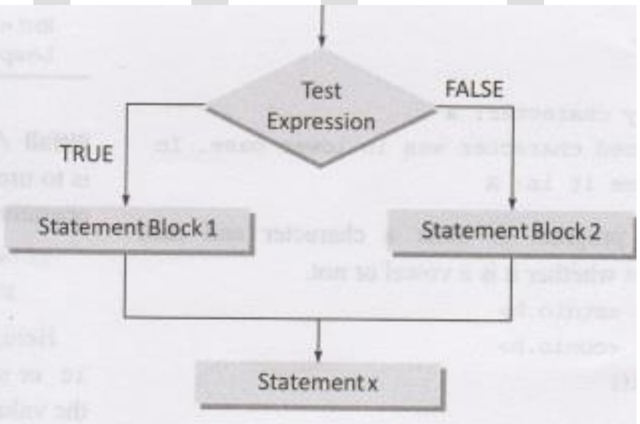


Figure 3.3 If-else statement construct

3. Write a program to find whether the given number is even or odd.

#include <stdio.h>

#include <conio.h>

```
int main()
{
int num;
clrscr();
printf("\n Enter any number: ");
scanf("%d", &num);
if (num%2 = = 0)
printf("\n %d is an even number", num);
else
printf("\n %d is an odd number", num);
return 0;
}
```

Output

```
Enter any number: 11
11 is an odd number
```

Write a program to enter a character and then determine whether it is a vowel or not.

```
#include <stdio.h>
int main() {
    char ch;

    printf("\n Enter any character: ");
    scanf("%c", & ch);
    if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||
        ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U') {
        printf("\n %c is a VOWEL", ch);
    } else {
        printf("\n %c is not a vowel", ch);
    }

    return 0;
```

Leap year or NOT

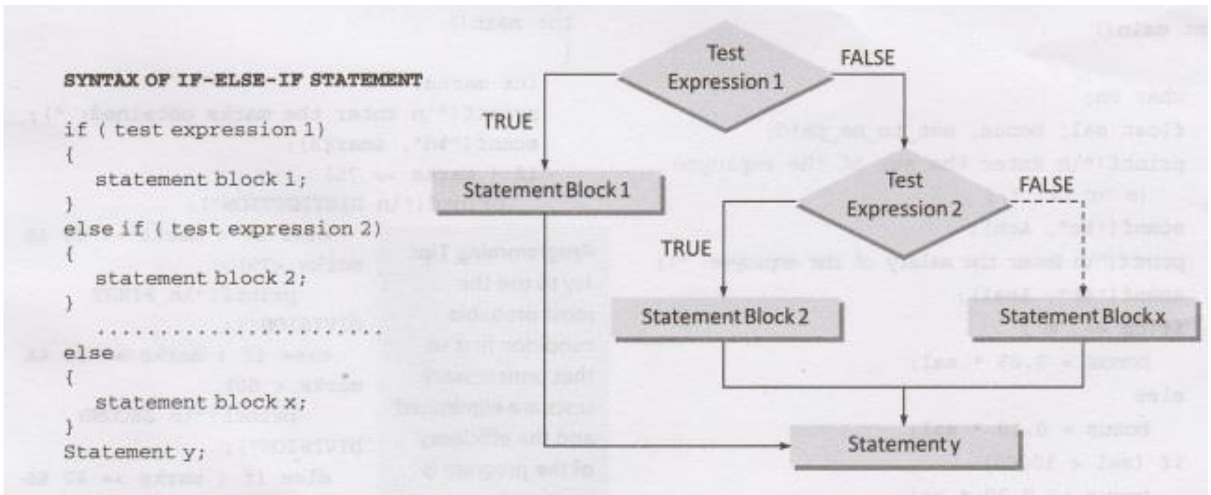
```
#include <stdio.h>
int main() {
    int year;

    printf("\n Enter any year: ");
    scanf("%d", &year);

    if ((year % 4 == 0) && ((year % 100 != 0) || (year % 400
== 0)))
        printf("\n Leap Year");
    else
        printf("\n Not A Leap Year");
    return 0;
}
```

If-Else-If Statement

C language supports if-else-if statements to test additional conditions apart from the initial test expression.



```
SYNTAX OF IF-ELSE-IF STATEMENT
if ( test expression 1)
{
    statement block 1;
}
else if ( test expression 2)
{
    statement block 2;
}
    .......................
else
{
    statement block x;
}
Statement y;
```

Figure 3.4 If–else–if statement construct

The *if-else-if* construct works in the same way as a normal if statement. *If-else-if* construct is also known as nested *if* construct. Its construct is given in Figure 3.4.

It is not necessary that every *if* statement should have an *else* block as C supports simple *if* statements. After the first test expression or the first if branch, the programmer can have as many *else-if* branches as he wants depending on the expressions that have to be tested. For example, the following code tests whether a number entered by the user is negative, positive, or equal to zero.

7. Write a program to demonstrate the use of nested if structure.

#include <stdio.h>

```
int main()
{
    int x, y;
    printf("\n Enter two numbers: ");
    scanf("%d %d", &x, &y);
    if (x == y)
        printf("\n The two numbers are equal");
    else if (x > y)
        printf("\n %d is greater than %d", x, y);
    else
        printf("\n %d is smaller than %d", x, y);
    return 0;
}
```

Output

```
Enter two numbers: 12 23
12 is smaller than 23
```

Write a program to test whether a number entered is positive, negative or equal to zero.

```
#include <stdio.h>


int main() {
    int num;
    printf("\n Enter any number: ");
    scanf("%d", &num);

    if (num == 0)
        printf("\n The number is equal to zero");
    else if (num > 0)
        printf("\n The number is positive");
    else
        printf("\n The number is negative");

    return 0;
}
```

Write a program to display the examination result.

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

    // Prompt user for input
    printf("\nEnter the marks obtained (0 to 100): ");
    scanf("%d", &marks);

    // Check if the input is within the valid range
    if (marks < 0 || marks > 100) {
        printf("\nInvalid marks! Please enter a value between 0
and 100.");
    } else {
        // Determine the division based on marks
        if (marks >= 75) {
            printf("\nDISTINCTION");
        } else if (marks >= 60) {
            printf("\nFIRST DIVISION");
        } else if (marks >= 50) {
            printf("\nSECOND DIVISION");
        } else if (marks >= 40) {
            printf("\nTHIRD DIVISION");
        } else {
            printf("\nFAIL");
        }
    }

    return 0;
}
```

Write a program to find the greatest of three numbers.

```
#include <stdio.h>
int main() {
    int num1, num2, num3;
    // Prompt user for input
    printf("Enter three numbers:\n");
    printf("Number 1: ");
    scanf("%d", &num1);
    printf("Number 2: ");
    scanf("%d", &num2);
    printf("Number 3: ");
    scanf("%d", &num3);

    // Determine the greatest number
    if (num1 >= num2 && num1 >= num3) {
        printf("The greatest number is: %d\n", num1);
    } else if (num2 >= num1 && num2 >= num3) {
        printf("The greatest number is: %d\n", num2);
    } else {
        printf("The greatest number is: %d\n", num3);
    }

    return 0;
}
```

Switch Case

A switch case statement is a multi-way decision statement that is a simplified version of an *if-else* block that evaluates only one variable. The general form of a *switch* statement is shown in Figure 3.5.

Programming Tip: ***It is always recommended to use default label in a switch statement.***

Table 3.1 compares general form of a *switch* statement with that of an *if-else* statement.

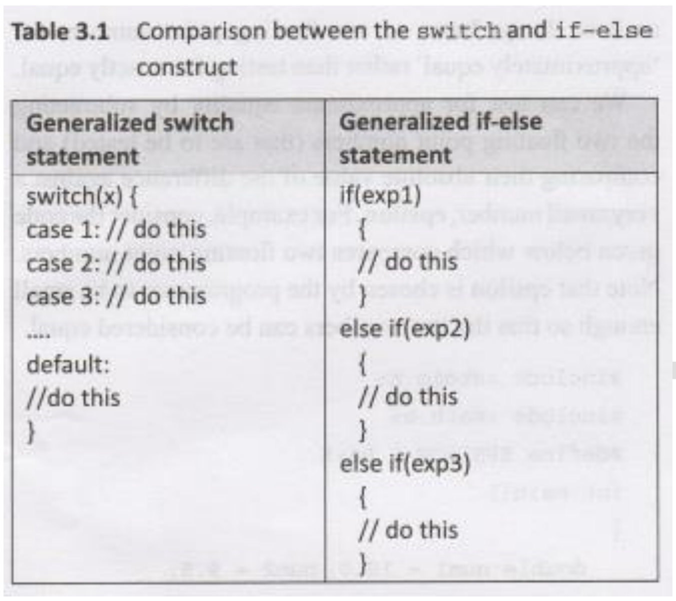


Table 3.1 Comparison between the `switch` and `if-else` construct

Generalized switch statement	Generalized if-else statement

switch(x) {
case 1: // do this
case 2: // do this
case 3: // do this
....
default:
//do this
}
	if(exp1)
	{
	// do this
	}
	else if(exp2)
	{
	// do this
	}
	else if(exp3)
	{
	// do this
	}

Here, statement blocks refer to statement lists that may contain zero or more statements. *These statements in the block are not enclosed within opening and closing braces.*

The power of nested *if-else* statements lies in the fact that it can evaluate more than one expression in a sin- gle logical structure. Switch statements are mostly used in two situations:

• When there is only one variable to evaluate in the expression

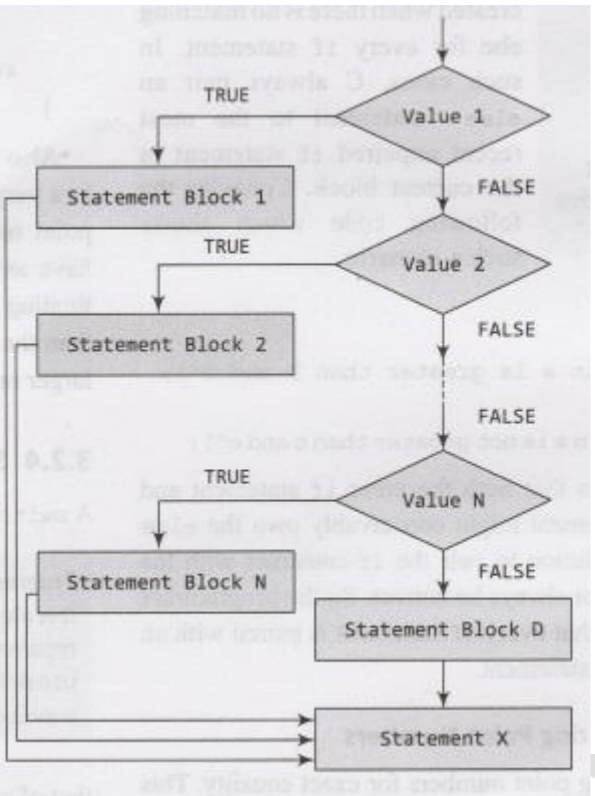
• When many conditions are being tested for

When there are many conditions to test, using the *if* and *else-if* construct becomes a bit complicated and confusing.

Therefore, *switch* case statements are often used as an alternative to long *if* statements that compare a variable to several *integral* values (integral values are those values that can be expressed as an integer, such as the value of a char). Switch statements are also used to handle the input given by the user.

We have already seen the syntax of the *switch* statement. The *switch* case statement compares the value of the variable given in the *switch* statement with the value of each case statement that follows. When the value of the *switch* and the *case* statement matches, the statement block of that particular case is executed.

Did you notice the keyword *default* in the syntax of the switch case statement? *Default* is also a case that is executed when the value of the variable does not match with any of the values of the *case* statement, i.e., the default case is executed when no match is found between the values of *switch* and *case* statements and thus there are no statements to be executed. Although the *default* case is optional, it is always recommended to include it as it handles any unexpected cases.



Syntax of Switch Statement

```
switch ( variable )
{
case  value1:
  Statement Block 1;
  break;
case  value2:
  Statement Block 2;
  break;
.......................... .
case  value N:
  Statement Block N;
  break;
default:
  Statement Block D;
  break;
}
Statement X;
```

Figure 3.5 The switch statement construct

In the syntax of the *switch* case statement, we have used another keyword *break*. The *break* statement must be used at the end of each case because if it were not used, then the case that matched and all the following cases will be executed.

Programming Tip: ***C supports decision control statements that can alter the flow of a sequence of instructions. A switch-case statement is a multi-way decision statement that is a simplified version of an if-else block that evaluates only one variable.***

For example, if the value of *switch* statement matched with that of *case 2*, then all the statements in *case 2* as well as rest of the cases including *default* will be executed.
The *break* statement tells the compiler to jump out of the switch case statement and execute the statement following the switch case construct. Thus, the keyword break is used to break out of the case statements. It indicates the end of a case

and prevents the program from falling through and executing the code in all the rest of the case statements.

Consider the following example of *switch* statement.

```
char grade = 'C';
switch (grade)
{
case '0':
printf("\n Outstanding");
break;
case 'A':
printf("\n Excellent");
break;
case 'B':
printf("\n Good");
break;
case 'C':
printf("\n Fair");
break;
case 'F':
printf("\n Fail");
break;
default:
printf("\n Invalid Grade");
break;
}
```

Output

```
Fair
```