Pemrograman Dasar

Week 6 : *Condition*

OUTLINES

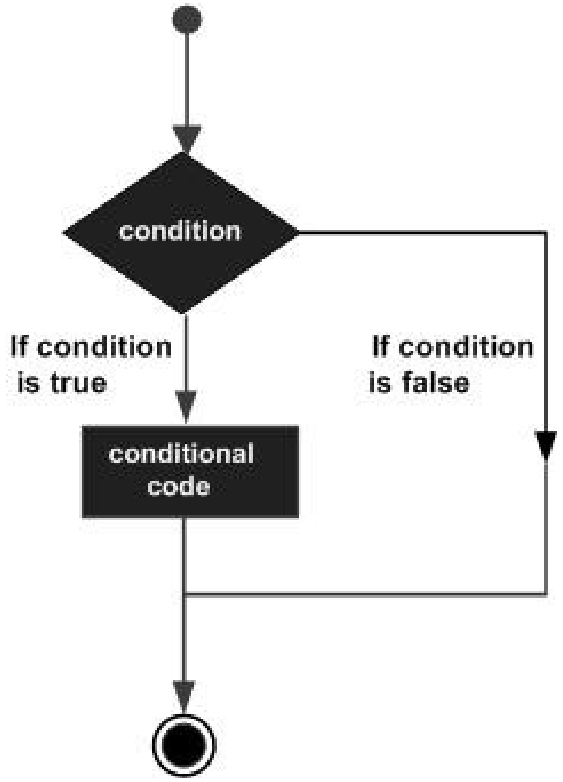
1. if statement
2. if…else statement
3. nested if statements
4. switch statement
5. nested switch statements
6. The ? : Operator

Decision making structures require that the programmer **specifies one or more conditions to be evaluated** or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Selection of this condition is very important in programming because with the condition selection, the **program can determine what process must be done next** based on the previous situation.

So it seems as if the program can think and make decisions. Herein lies the lack of a computer that is not able to think on its own, **all things done are based on orders**.

Show below is the general form of a typical decision making structure found in most of the programming languages



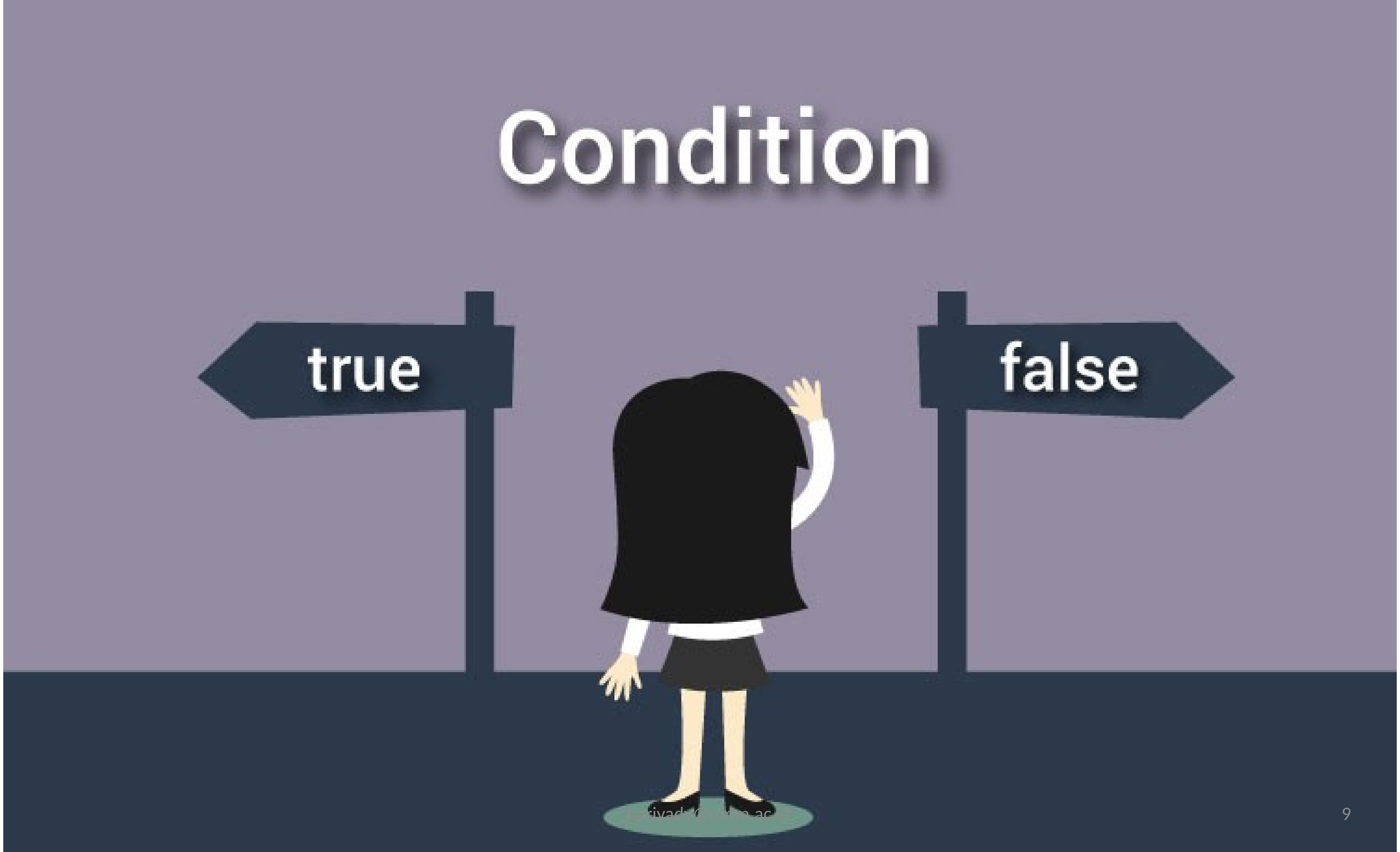
C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

ProgDas6-1.2.c

```
#include <stdio.h>

int main()
{
  int x=1;
  int y=2;

  int nilaiBetul = x<y;
  int nilaiSalah = x>y;

  printf("nilaiBetul %d\n", nilaiBetul);
  printf("nilaiSalah %d\n", nilaiSalah);

  return 0;
}
```

The test condition is generally expressed with the help of the relational operators in C. We have already seen relational operators in the previous chapter. To revise :

Operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to


Condition
true
false

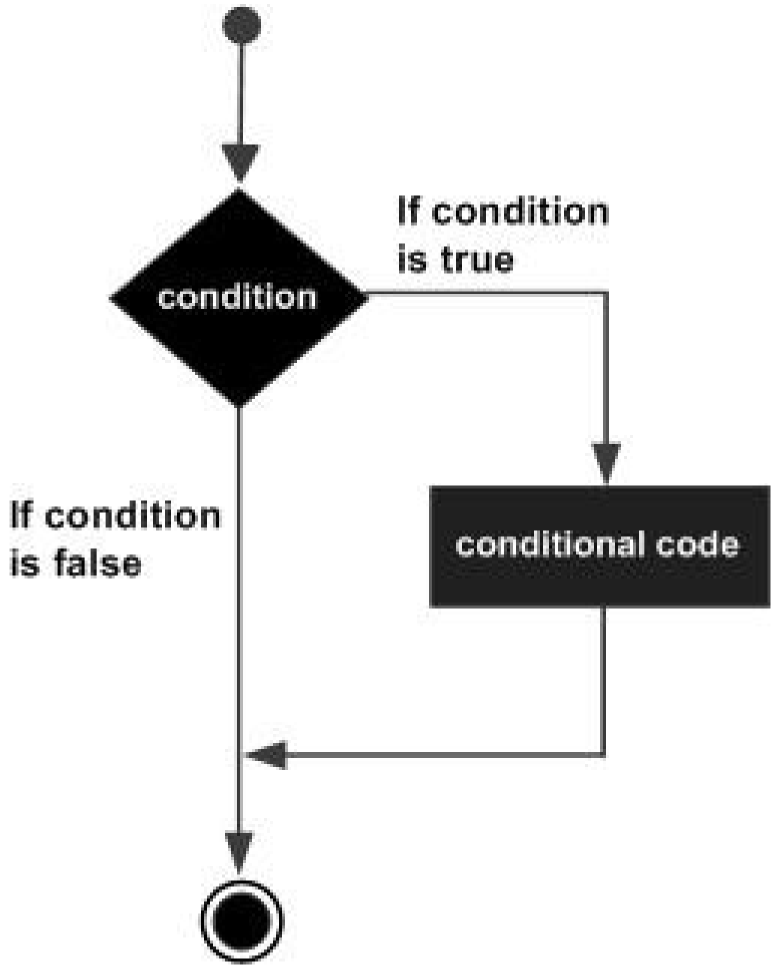
1. if statement

The syntax of an 'if' statement in C programming language is –

```
if(boolean_expression) {
   /* statement(s) will execute if the boolean
expression is true */
}
```

If the Boolean expression evaluates to true, then the block of code inside the 'if' statement will be executed. If the Boolean expression evaluates to false, then the first set of code after the end of the 'if' statement (after the closing curly brace) will be executed.

Flow Diagram



ProgDas6-1.c

```
#include <stdio.h>

int main ()
{
  int a = 10;
  if( a < 20 )
  {
  printf("a is less than 20\n" );
  }
  printf("value of a is : %d\n", a);
  return 0;
}
```

EXERCISE 1

Create a program to identify the highest value of the 3 values entered by the user!

Exercise6-1.c

2. if…else statement

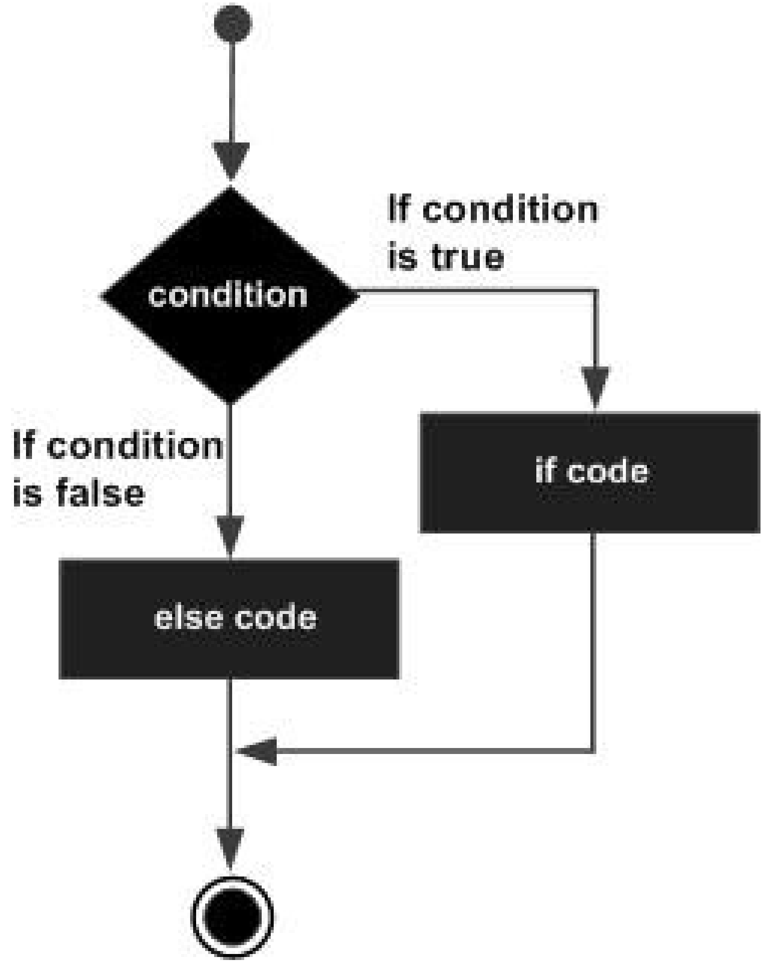
An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

The syntax of an if...else statement in C programming language is −

```
if(boolean_expression) {
   /* statement(s) will execute if the boolean expression is
true */
} else {
   /* statement(s) will execute if the boolean expression is
false */
}
```

If the Boolean expression evaluates to true, then the if block will be executed, otherwise, the else block will be executed.

Flow Diagram



ProgDas6-2.c

EXERCISE 2

Create a program to identify whether the number entered by the user is an even or odd number

Exercise6-2.c

If...else if...else Statement

An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

Once an else if succeeds, none of the remaining else if's or else's will be tested.

```
if(boolean_expression 1) {
   /* Executes when the boolean expression 1 is true */
} else if( boolean_expression 2) {
   /* Executes when the boolean expression 2 is true */
} else if( boolean_expression 3) {
   /* Executes when the boolean expression 3 is true */
} else {
   /* executes when the none of the above condition is
true */
}
```

ProgDas6-3.c

EXERCISE 3

Create a program to identify student grades.

if the value is < 60 then E

if the value is < 70 then D

if the value is < 80 then C

if the value is < 90 then B

if the value <= 100 then A

Exercise6-3.c

3. nested if statements

```
if( boolean_expression 1) {

   /* Executes when the boolean expression 1 is true */
   if(boolean_expression 2) {
      /* Executes when the boolean expression 2 is true */
   }
}
```

You can nest else if...else in the similar way as you have nested if statements.

ProgDas6-4.c

4. switch statement

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

case
case
case
case

```
switch(expression) {

   case constant-expression  :
      statement(s);
      break; /* optional */

   case constant-expression  :
      statement(s);
      break; /* optional */

   /* you can have any number of case statements */
   default : /* Optional */
   statement(s);
}
```

The following rules apply to a switch statement −

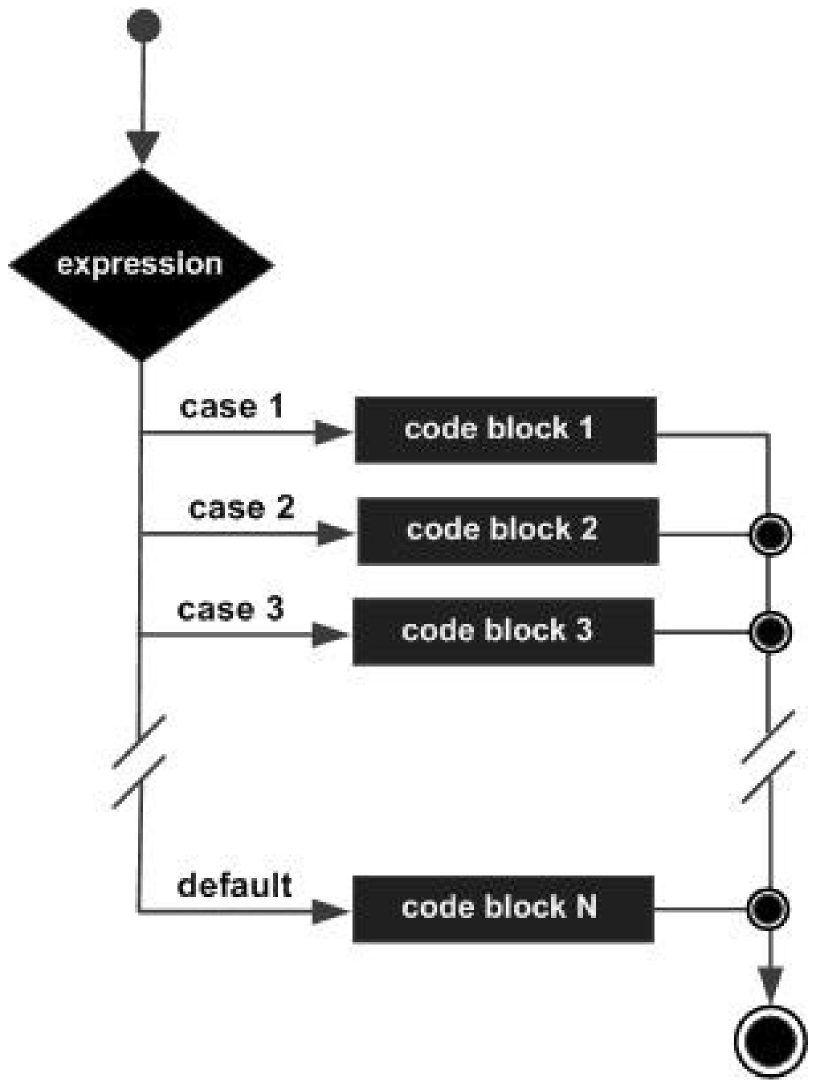
- The expression used in a switch statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.

- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.

- Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.

- A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Flow Diagram



ProgDas6-5.c

Important Points about Switch Case

1. The expression provided in the switch should result in a constant value otherwise it would not be valid.

Valid expressions for switch:

```
// Constant expressions allowed
switch(1+2+23)
switch(1*2+3%4)
```

Invalid switch expressions for switch:

```
// Variable expression not allowed
switch(ab+cd)
switch(a+b+c)
```

Important Points about Switch Case (2)

2. Duplicate case values are not allowed.

3. The default statement is optional.Even if the switch case statement do not have a default statement, it would run without any problem.

4. The break statement is used inside the switch to terminate a statement sequence. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.

5. The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.

6. Nesting of switch statements are allowed, which means you can have switch statements inside another switch. However nested switch statements should be avoided as it makes program more complex and less readable.

5. nested switch statements

It is possible to have a switch as a part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

```
switch(ch1) {

   case 'A':
      printf("This A is part of outer switch" );

      switch(ch2) {
         case 'A':
            printf("This A is part of inner switch" );
            break;
         case 'B': /* case code */
      }

      break;
   case 'B': /* case code */
}
```

ProgDas6-6.c

6. The ? : Operator

We have covered conditional operator ? :, or is also called ternary, in the previous chapter which can be used to replace if…else statements. It has the following general form –

Syntax : `Exp1 ? Exp2 : Exp3;`

Example : `(A > 100 ? 0 : 1);`

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

The value of a ? expression is determined like this −

- Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire expression.
- If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the entire expression.

ProgDas6-7.c

```
#include <stdio.h>

int main()
{
    int x=1, y ;
    y = ( x ==1 ? 2 : 0 ) ;
    printf("x value is %d\n", x);
    printf("y value is %d", y);
}
```

THANKYOU