Ministry of Higher Education & Scientific Research

PAITAXT

Technical Institute-Private

1st Year Computer & Network

C++ Programming

Chapter 3:

Control Structures (Selection)

Updated by: ***Asst. Prof. Dr. Tahseen G. Abdullah***

Control Structures

- A computer program can proceed:
  - In sequence
  - Selectively (branch) - making a choice
  - Repetitively (iteratively) - looping
- Some statements are executed only if certain conditions are met
- A condition is met if it evaluates to `true`
- A condition is represented by a logical (Boolean) expression that can be `true` or `false`
- Relational operators:
  - Allow comparisons
  - Require two operands (binary)
  - Evaluate to `true` or `false`

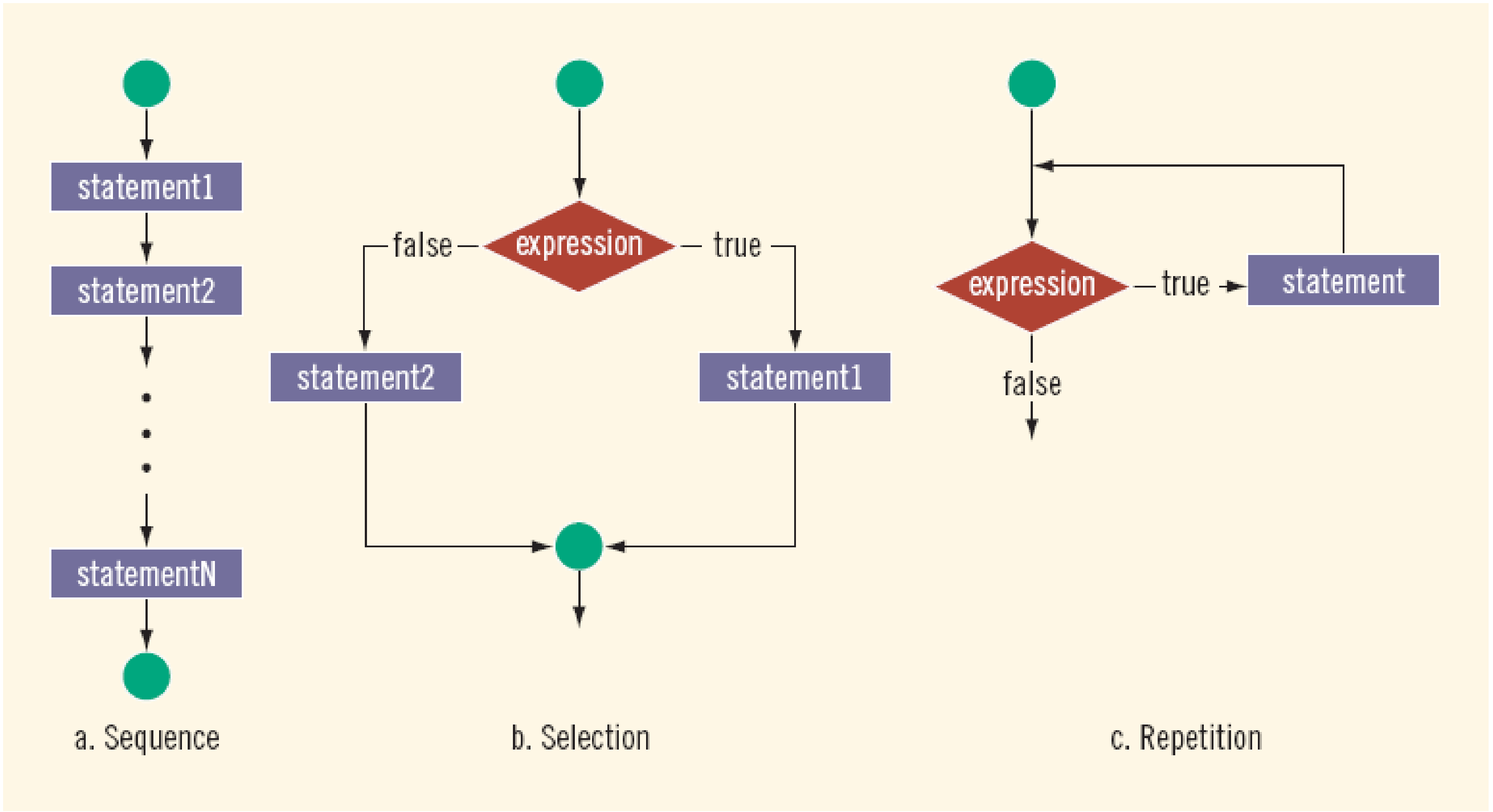


Fig.: Flow of execution.

Relational Operators

Operator	Description	Example
==	Equal to	`6 == 6` evaluates to `true`
!=	Not equal to	`6 != 6` evaluates to `false`
<	Less than	`8 < 15` evaluates to `true`
<=	Less than or Equal to	`5.9 <= 7.5` evaluates to `true`
>	Greater than	`2.5 > 5.8` evaluates to `false`
>=	Greater than or Equal to	`5.9 >= 7.5` evaluates to `false`

- You can use the relational operators with all three simple data types:
- Relational operators can be applied to strings:
  - 'A' > 'B' evaluates to `false`
  - !('A' > 'B') evaluates to `true`
  - "Hello" < "Hi" evaluates to `true`
  - "Hello" > "Hen" evaluates to `false`
  - 'a' > 'B' evaluates to `true`

  - Note: `true` has the value 1 and `false` has the value 0.

Example Program: (Relational and Boolean Operators)

```
 #include <iostream>
 #include <string>
 using namespace std;
 int main()
   {
int n=2,m=5;
char ch1='a', ch2='A', ch3='B';
string str1= "Hi", str2= "Hello";
cout <<     m                 << endl;
cout <<    (n+6)              << endl;
cout << (n+m/2)               << endl;
cout <<   (n>10)              << endl;
cout <<  (m<10)               << endl;
cout << (ch1<ch2)             << endl;
cout << (ch3>ch2)             << endl;
cout << (str1>str2)           << endl;
cout << str1<< "  \&  " <<str2<< endl;
return 0;
  }
```

Output Results:

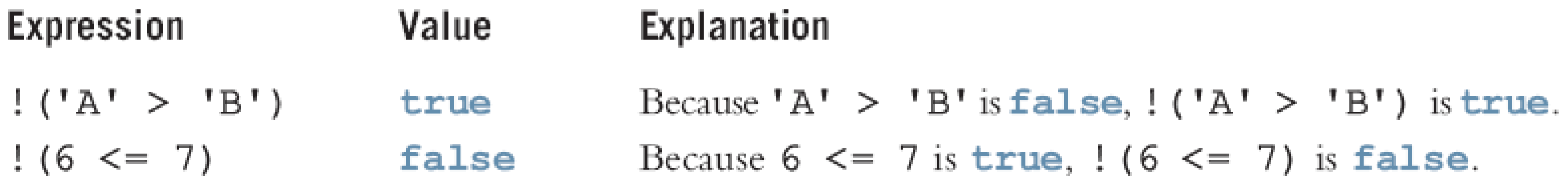
- 5
- 8
- 4
- 0
- 1
- 0
- 1
- 1
- Hi & Hello

Logical (Boolean) Operators (! , && , | |)

- **not operator (!)**

Expression	!(Expression)
`true`	`false`
`false`	`true`

Example

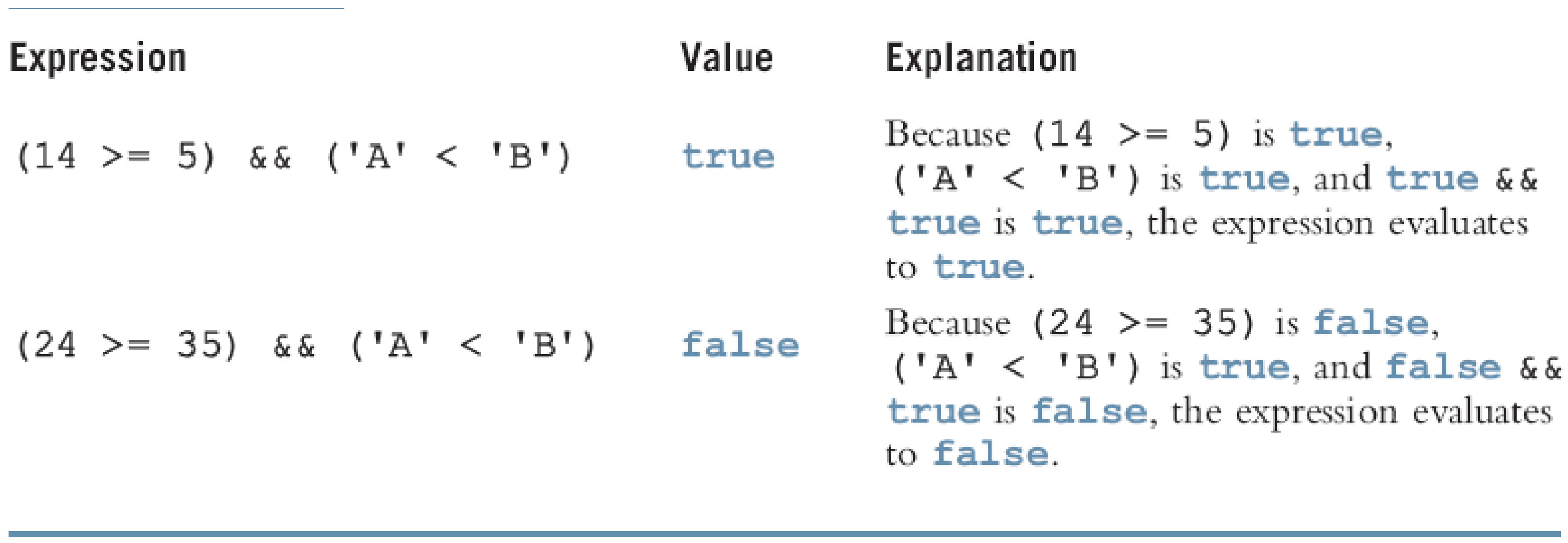


Expression	Value	Explanation
`!('A' > 'B')`	`true`	Because `'A' > 'B'` is `false`, `!('A' > 'B')` is `true`.
`!(6 <= 7)`	`false`	Because `6 <= 7` is `true`, `!(6 <= 7)` is `false`.

- Logical expressions evaluate to either 1 or 0
- You can use the `int` data type to manipulate logical (Boolean) expressions
- The data type `bool` has logical (Boolean) values `true` and `false`
  - The identifier `true` has the value 1
  - The identifier `false` has the value 0

▪ and operator (&&)

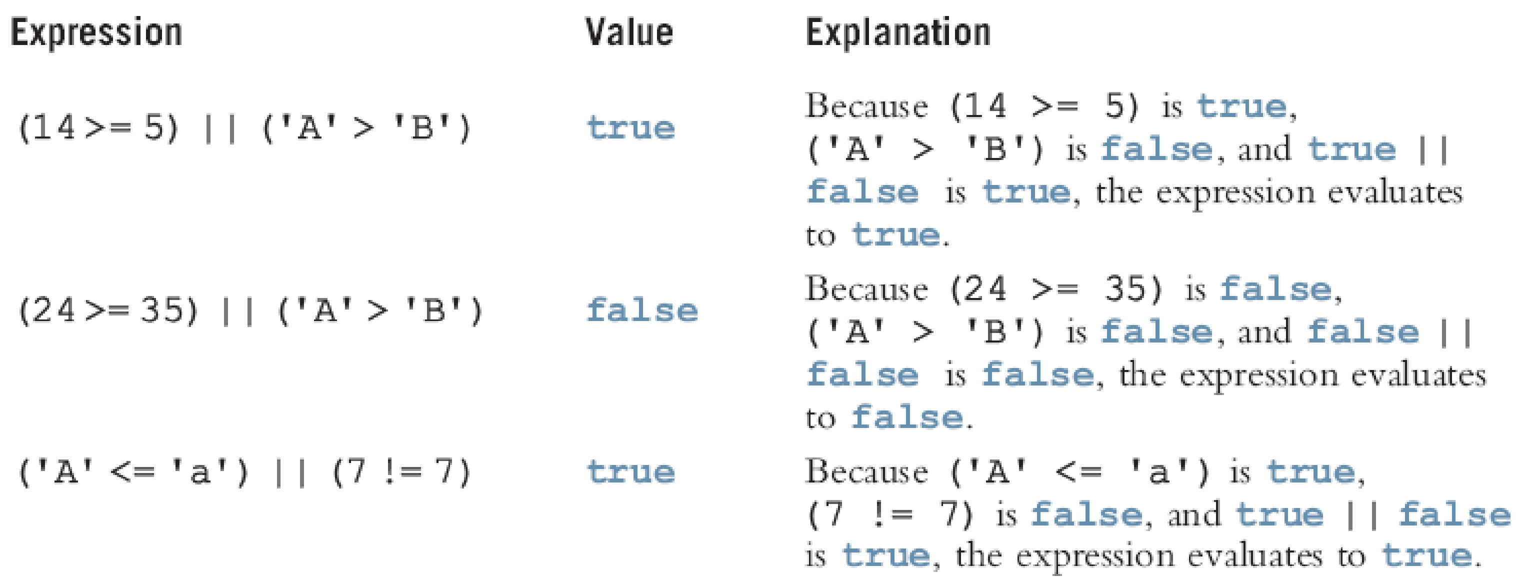
Expression	Expression	Expression **&&** Expression
`true`	`true`	`true`
`true`	`false`	`false`
`false`	`true`	`false`
`false`	`false`	`false`



Expression	Value	Explanation
`(14 >= 5) && ('A' < 'B')`	`true`	Because `(14 >= 5)` is `true`, `('A' < 'B')` is `true`, and `true && true` is `true`, the expression evaluates to `true`.
`(24 >= 35) && ('A' < 'B')`	`false`	Because `(24 >= 35)` is `false`, `('A' < 'B')` is `true`, and `false && true` is `false`, the expression evaluates to `false`.

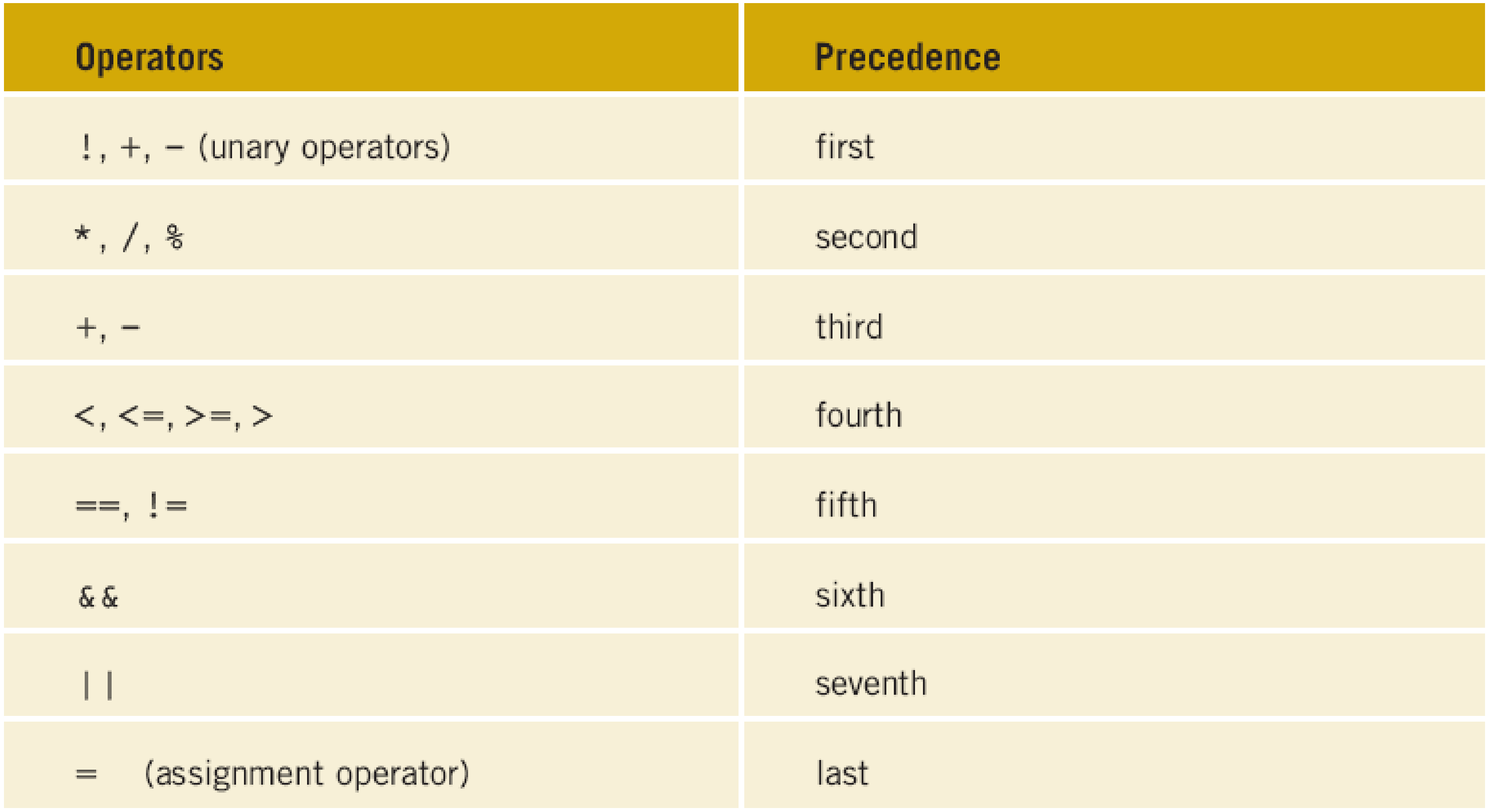
▪ or operator (||)

Expression	Expression	Expression || Expression
`true`	`true`	`true`
`true`	`false`	`true`
`false`	`true`	`true`
`false`	`false`	`false`



Expression	Value	Explanation
`(14 >= 5) || ('A' > 'B')`	`true`	Because `(14 >= 5)` is `true`, `('A' > 'B')` is `false`, and `true || false` is `true`, the expression evaluates to `true`.
`(24 >= 35) || ('A' > 'B')`	`false`	Because `(24 >= 35)` is `false`, `('A' > 'B')` is `false`, and `false || false` is `false`, the expression evaluates to `false`.
`('A' <= 'a') || (7 != 7)`	`true`	Because `('A' <= 'a')` is `true`, `(7 != 7)` is `false`, and `true || false` is `true`, the expression evaluates to `true`.

Order of Precedence



Operators	Precedence
!, +, - (unary operators)	first
*, /, %	second
+, -	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

Q.: Which of the following has the last precedence in C++?

(a) && **(b) !** **(c) =** **(d) ||**

Example Program: (Relational and Boolean Operators)

```
     #include <iostream>
      using namespace std;
      int main()
       {
int n,m;
cout << (n=5) << endl;
cout << (n == 5) << endl;
cout << (n > 3) << endl;
cout << (n < 4) << endl;
cout << (m = 10) << endl;
cout << (m== 0) << endl;
cout << (m > 0) << endl;
cout << (m< 4 && n==5) << endl;
cout << (m<4 || n==5) << endl;
cout << (!m) << endl;
cout << ('a'>'A') << endl;
      return 0;
}
```

Output Results:

- 5
- 1
- 1
- 0
- 10
- 0
- 1
- 0
- 1
- 0
- 1

Example:

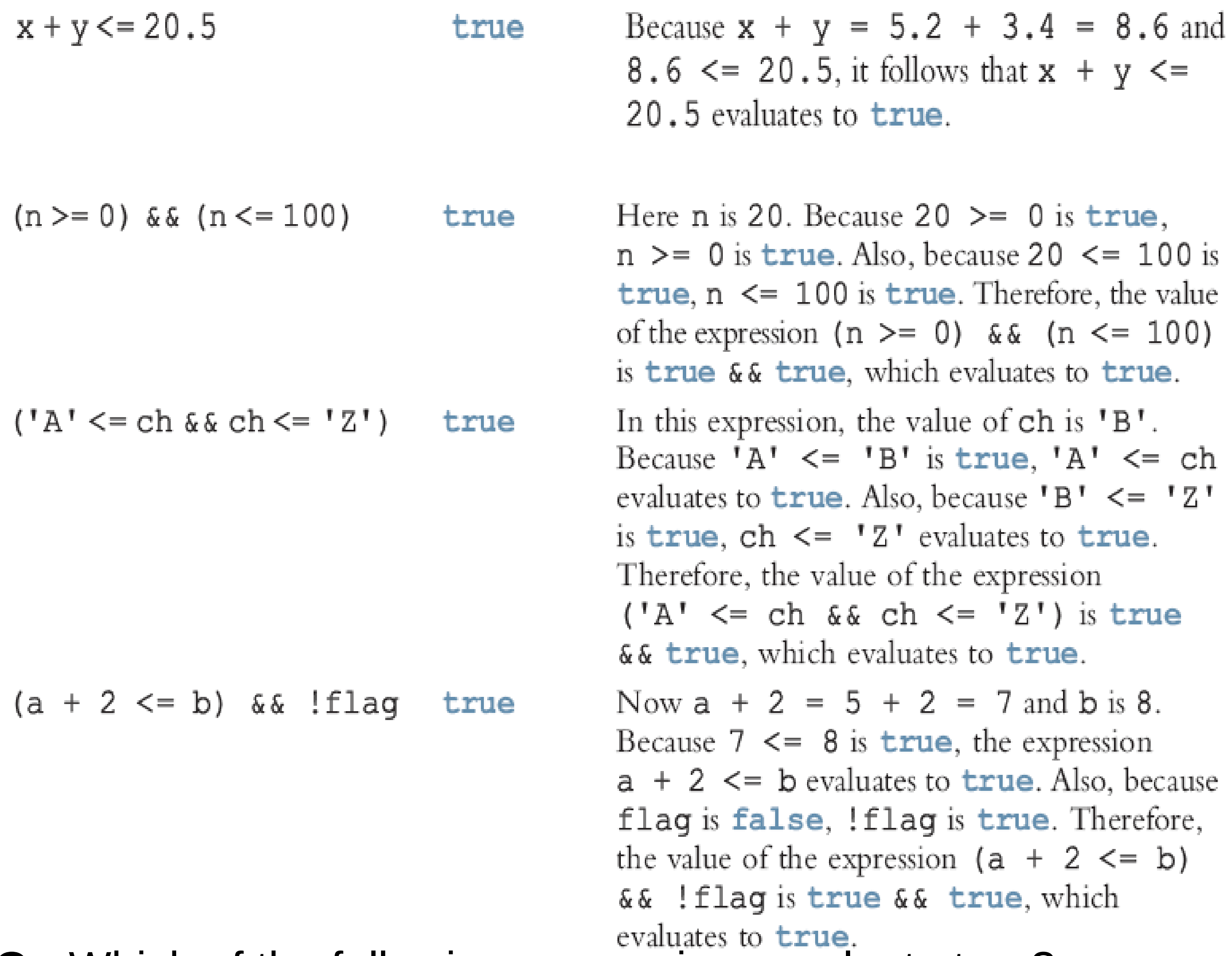
Suppose you have the following declarations:

```
bool found = true;
bool flag = false;
int num = 1;
double x = 5.2;
double y = 3.4;
int a = 5, b = 8;
int n = 20;
char ch = 'B';
```

Expression	Value	Explanation
`!found`	`false`	Because `found` is `true`, `!found` is `false`.
`x > 4.0`	`true`	Because `x` is `5.2` and `5.2 > 4.0` is `true`, the expression `x > 4.0` evaluates to `true`.
`!num`	`false`	Because `num` is `1`, which is nonzero, `num` is `true` and so `!num` is `false`.
`!found && (x >= 0)`	`false`	In this expression, `!found` is `false`. Also, because `x` is `5.2` and `5.2 >= 0` is `true`, `x >= 0` is `true`. Therefore, the value of the expression `!found && (x >= 0)` is `false && true`, which evaluates to `false`.
`!(found && (x >= 0))`	`false`	In this expression, `found && (x >= 0)` is `true && true`, which evaluates to `true`. Therefore, the value of the expression `!(found && (x >= 0))` is `!true`, which evaluates to `false`.

Q.: Which of the following data type has logical (Boolean) values `true` and `false`?

(a) int **(b) double** **(c) char** **(d) bool**



`x + y <= 20.5`	`true`	Because `x + y = 5.2 + 3.4 = 8.6` and `8.6 <= 20.5`, it follows that `x + y <= 20.5` evaluates to `true`.
`(n >= 0) && (n <= 100)`	`true`	Here `n` is `20`. Because `20 >= 0` is `true`, `n >= 0` is `true`. Also, because `20 <= 100` is `true`, `n <= 100` is `true`. Therefore, the value of the expression `(n >= 0) && (n <= 100)` is `true && true`, which evaluates to `true`.
`('A' <= ch && ch <= 'Z')`	`true`	In this expression, the value of `ch` is `'B'`. Because `'A' <= 'B'` is `true`, `'A' <= ch` evaluates to `true`. Also, because `'B' <= 'Z'` is `true`, `ch <= 'Z'` evaluates to `true`. Therefore, the value of the expression `('A' <= ch && ch <= 'Z')` is `true && true`, which evaluates to `true`.
`(a + 2 <= b) && !flag`	`true`	Now `a + 2 = 5 + 2 = 7` and `b` is `8`. Because `7 <= 8` is `true`, the expression `a + 2 <= b` evaluates to `true`. Also, because `flag` is `false`, `!flag` is `true`. Therefore, the value of the expression `(a + 2 <= b) && !flag` is `true && true`, which evaluates to `true`.

Q.: Which of the following expressions evaluate true?

(a) **(b)** **(c)** **(d)**

Example Program: (Relational and Boolean Operators)

```
   #include <iostream>
   using namespace std;
   int main()
      {
bool found=true;
bool flag=false;
int num=1, a=5, b=8, n=20;
double x=5.2, y=3.4;
char ch='B';
cout << (!found) << endl;
cout << (x>4.0) << endl;
cout << (!num) << endl;
cout << (!found && x>=0) << endl;
cout << !(found && x>=0) << endl;
cout << (x+y<=20.5) << endl;
cout << (a+2<=b && !flag) << endl;
cout << (ch== ' b') << endl;
  return 0;
}
```

Output Results:

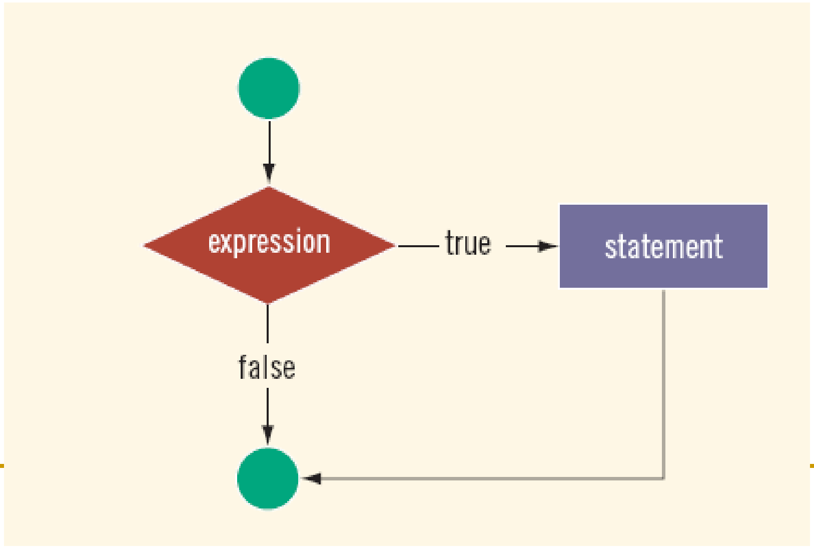
- 0
- 1
- 0
- 0
- 0
- 1
- 1
- 0

if statement (One-Way Selection)

- The syntax of one-way selection is:

```
if (expression)
    statement
```

- There is no a semicolon at the end of `if` statement
- The statement is executed if the value of the expression is `true`
- The statement is bypassed if the value is `false`; program goes to the next statement
- `if` is a reserved word



Example Program: (Using if statement)

```
// Using if statement to know that ( a person eligible to vote or not)
 #include <iostream>
 using namespace std;
 int main()
 {
   int age;
   cout << "Enter the age\n";
   cin >> age;

  if (age>=18)
  cout << " Eligible to vote." << endl;
  if (age<18)
  cout << " Not Eligible to vote." << endl;
 return 0;
 }
```

Example Program: (Using if statement)

```
// if statement for determining the grade of the score
#include <iostream>
using namespace std;
int main()
{
   int score;

   cout << "Enter the score \n";
   cin >> score;

  if (score>=50)
  cout << " The grade is \  PASS " << endl;
  if (score<50)
  cout << " The grade is \  FAIL" << endl;

 return 0;
}
```

Output Results:

```
Enter the score
45
The grade is  FAIL
```

Output Results:

```
Enter the score
75
The grade is  PASS
```

Assignment (H.W)

- Assume that score is a variable of type integer. Based on the value of the score in the table. Determine the outputs of the grade using if statement.

score	grade
>=90	A
>=80	B
>=70	C
>=60	D
<60	F

Compound (Block of) Statement

```
{
    statement1
    statement2
        .
        .
        .
    statementn
}
```

- A compound statement (block of statement) is a single statement:

Example

```
if (age >= 18)
{
  cout << "Eligible to vote." << endl;
  cout << "No longer a minor." << endl;
}
if (age < 18)
{
  cout << "Not eligible to vote." << endl;
  cout << "Still a minor." << endl;
}
```

Q.: What notation is used to place compound (block of) statement in C++?

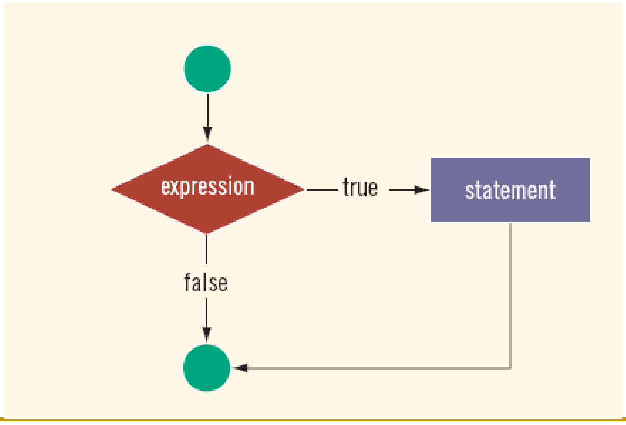
(a) << >> **(b) ( )** **(c) { }** **(d) [ ]**

if-else statement (Two-Way Selection)

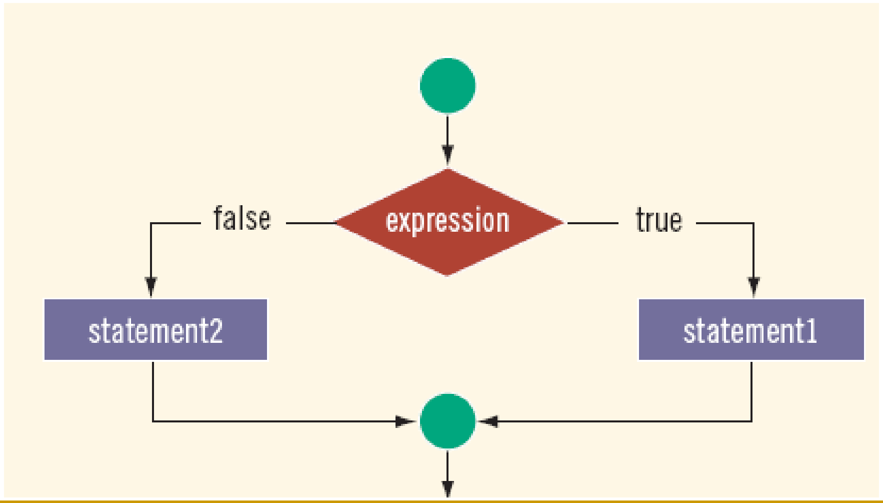
- Two-way selection takes the form:

```
if (expression)
     statement1
else
     statement2
```

- If expression is `true`, `statement1` is executed; otherwise, `statement2` is executed
  - `statement1 and statement2 are any C++ statements`
- `else` is a reserved word



One-Way Selection



Two-Way Selection

Example Program: (Using if-else statement)

```
// Using if-else statement to know that ( a person eligible to vote or not)
#include <iostream>
using namespace std;
int main()
{
   int age;
   cout << "Enter the age\n";
   cin >> age;

  if (age>=18)
  cout << "Eligible to vote." << endl;
  else
  cout << “Not Eligible to vote." << endl;
 return 0;
}
```

Example Program: (Using if –else Statement)

```
// if-else statement for determining the grade of the score
#include <iostream>
using namespace std;
int main()
{
   int score;
   char grade;
   cout << "Enter the score \n";
   cin >> score;

    if (score>=50)
    grade='P';
    else
    grade='F';
    cout << “The grade is    " << grade<< endl;
   return 0;
}
```

Output Results:

```
Enter the score
45
The grade is F
```

Output Results:

```
Enter the score
75
The grade is P
```

Multiple Selections: Nested if

- **Example:** Assume that score is a variable of type int. Based on the value of score determine the outputs of the grade.

```
#include <iostream>
using namespace std;
int main()
{
   int score;
             cout << "Enter the score \n";
   cin >> score;
           if (score>=90)
            cout << "The grade is  A. " << endl;
    else if (score>=80)
            cout << "The grade is  B. " << endl;
    else if (score>=70)
            cout << "The grade is  C." << endl;
    else if (score>=60)
            cout << "The grade is  D." << endl;
    else
            cout << "The grade is  F." << endl;
   return 0;
}
```

score	grade
>=90	A
>=80	B
>=70	C
>=60	D
<60	F

Example Program: (Using if-else statement) (H.W)

- Write a C++ program to determine the names of the months according to the following table.

Month	Name
1	January
2	February
3	March
4	April
5	May
6	June
7	July
8	August
9	September
10	October
11	November
12	December

Switch Structures

- `switch` structure: alternate to if-else
- `Switch` (integral) expression is evaluated first
- Value of the expression determines which corresponding action is taken
- Expression is sometimes called the selector
- One or more statements may follow a case label
- The `break` statement may or may not appear after each statement
- `switch`, `case`, `break`, and `default` are reserved words

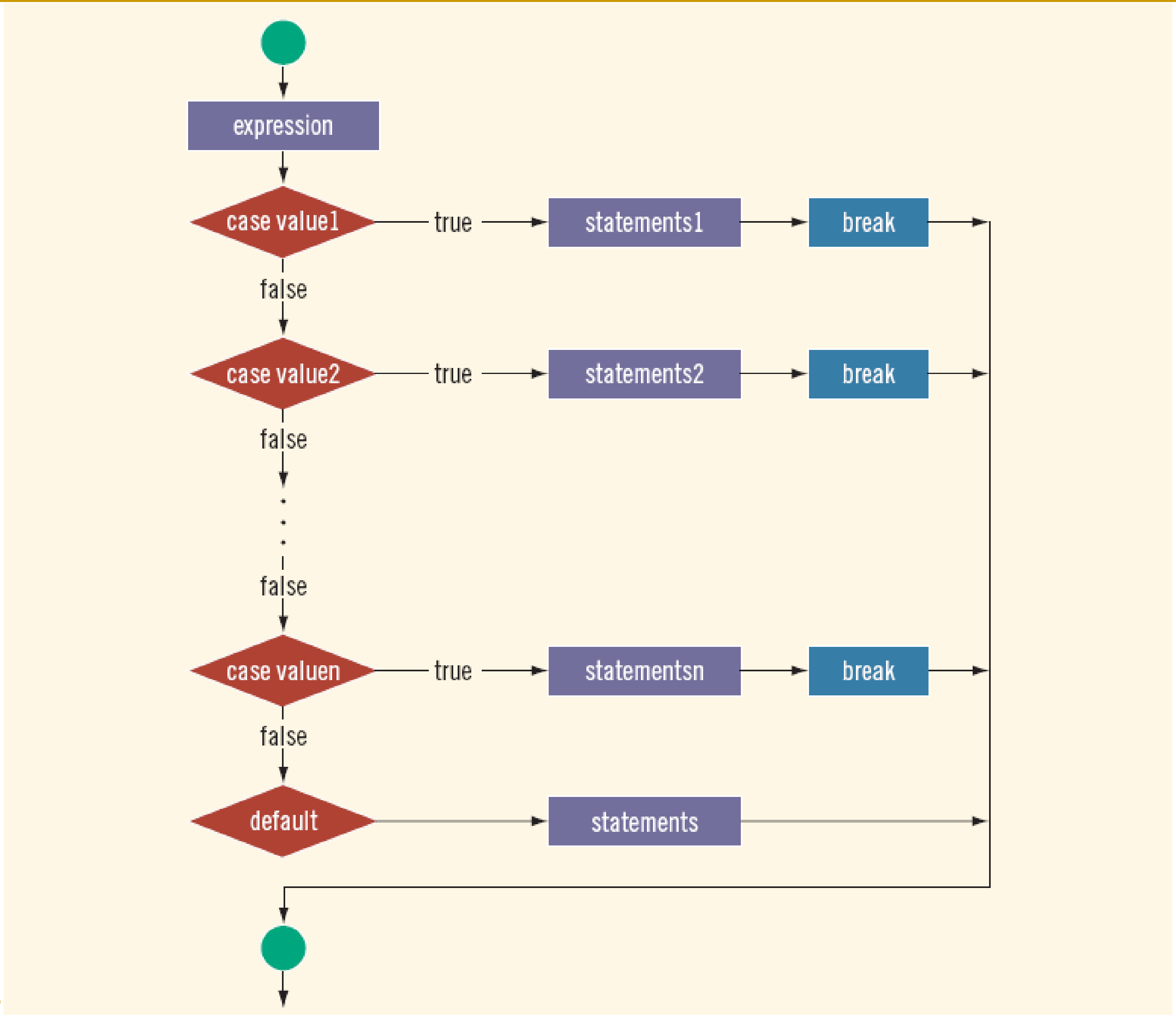
```
switch (expression)
{
case value1:
         statements1
         break;
case value2:
         statements2
         break;
         .
         .
         .
case valuen:
         statementsn
         break;
default:
         statements
}
```

Q.: Which follows the case statement in C++?

(a) ; **(b) .** **(c) :** **(d) ,**

Ex. (H.W):

Solve Problem Page 23 Using Switch Structures


expression
case value1
true
statements1
break
false
case value2
true
statements2
break
false
false
case valuen
true
statementsn
break
false
default
statements

Example Program: (using switch structure)

- Write a C++ program to determine the grade according to the following character.

grade character	grade number
A	4.0
B	3.0
C	2.0
D	1.0
F	0.0

```
#include <iostream>
 using namespace std;
 int main()
{
  char grade;
  cout<<"Enter the Character"<<endl;
  cin>>grade;
 switch (grade)
 {
case ' A '  :
    cout << "The grade is  4.0 " << endl;
    break;
case ' B '  :
    cout << "The grade is  3.0 " << endl;
    break;
case ' C '  :
    cout << "The grade is  2.0 " << endl;
    break;
case ' D '  :
    cout << "The grade is  1.0 " << endl;
    break;
case ' F '  :
    cout << "The grade is  4.0 " << endl;
    break;
default :
    cout << “No grade  " << endl;
    }
 return 0;
 }
```

<u>Ex. (H.W):</u> P.23 Switch Structures

End of the Lecture

Let Learning Continue

Thank You