Hochiminh City University of Technology
Computer Science and Engineering
[CO1011 - 501127] - Fundamentals of C++ Programming

Control Flow - If

Lecturer: Dr. Duc Dung Nguyen
Dr. Rang Nguyen

Credits: 4

Outcomes

- Understand basic control structures in C/C++
  - if-else statement
  - switch statement
- Solve the problem using conditional executions
- Implement if-else, switch-case statements

Today's outline

- Relational and logical operators
- if-else statement
    - Nested conditionals
- switch statement
    - Enum type

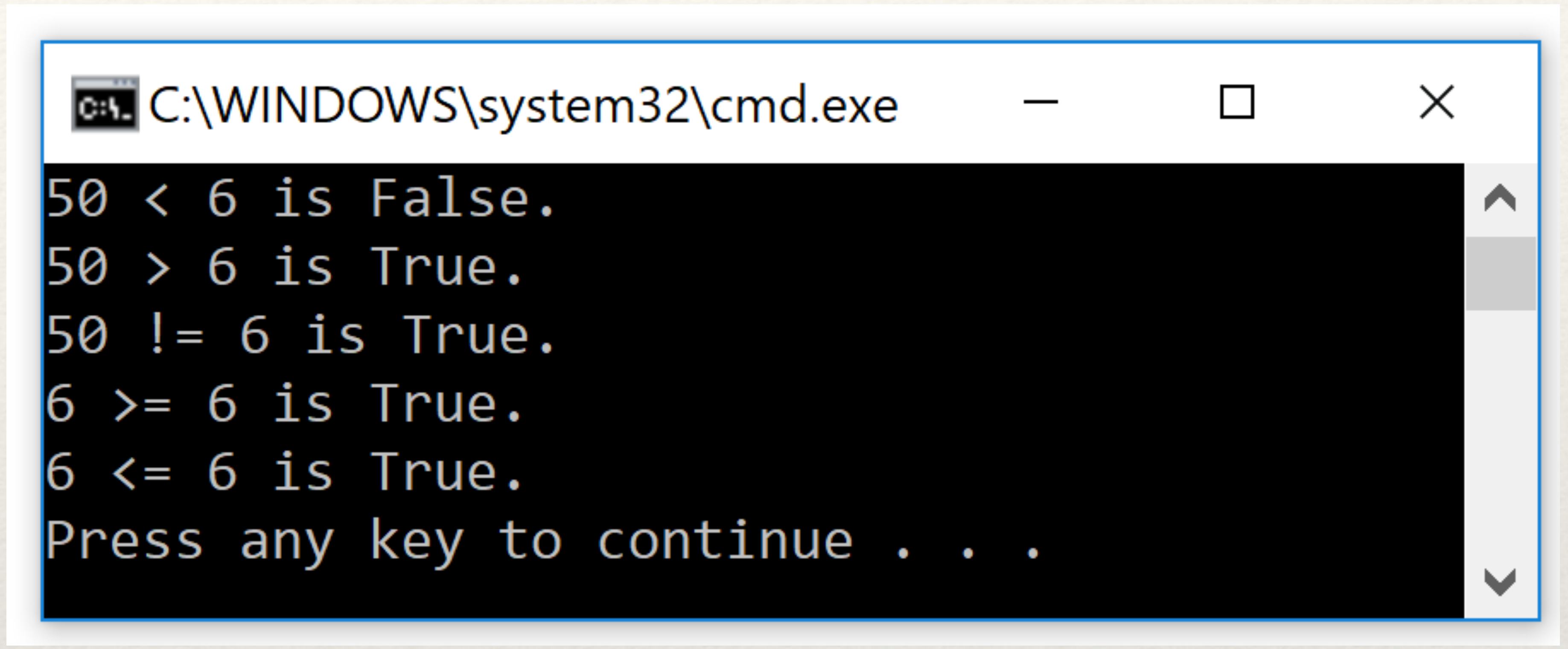
Relational and logical operators

Relational operators

Operator	Meaning
“=="	Equal to
“<"	Less than
“>"	Greater than
“<="	Less than or equal to
“>="	Greater than or equal to
“ != ”	Not equal to

Relational operators

```cpp
#include<iostream>
using namespace std;

int main() {
    //instead of printing 0 and 1, create an array where
    //0 = False, 1 = True
    string TorF[] = { "False", "True" };
    int a = 50, b = 6, c = 6;

    //Print out the string values of each relational operation
    printf("%d < %d is %s.\n", a, b, TorF[a < b].c_str());
    printf("%d > %d is %s.\n", a, b, TorF[a > b].c_str());
    printf("%d != %d is %s.\n", a, b, TorF[a != b].c_str());
    printf("%d >= %d is %s.\n", b, c, TorF[b >= c].c_str());
    printf("%d <= %d is %s.\n", b, c, TorF[b <= c].c_str());
    return 0;
}
```



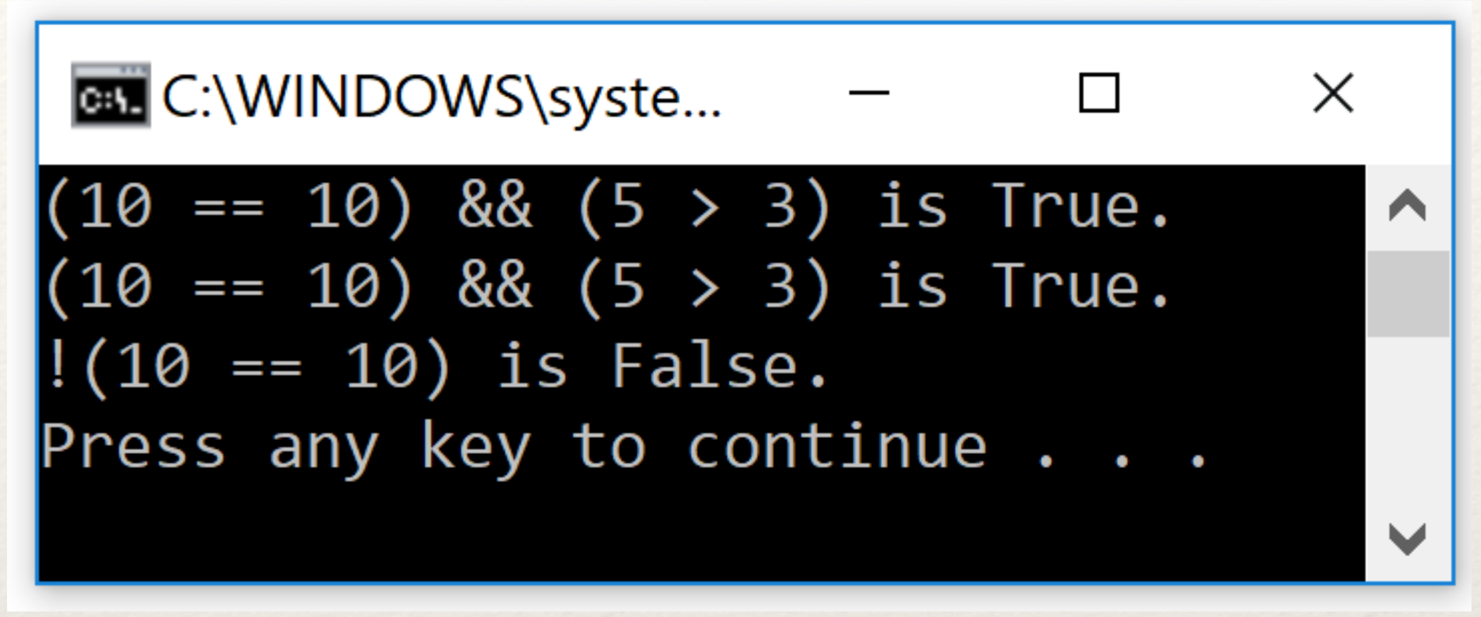
```
50 < 6 is False.
50 > 6 is True.
50 != 6 is True.
6 >= 6 is True.
6 <= 6 is True.
Press any key to continue . . .
```

Logical operators

Operator	Meaning	Behavior
&&	and	If both inputs are true the outcome of the operation is true; otherwise false
| |	or	If both inputs are false the outcome of the operation is false; otherwise true
!	not	Negates the logical condition

Logical operators

```
#include<iostream>
using namespace std;

int main() {
  int a = 10, b = 5, c = 10, d = 3;
  std::string TorF[] = { "False", "True" };

  //The && operator
  printf("(%d == %d) && (%d > %d) is %s.\n", a, c, b, d, TorF[(a == c) && (b > d)].c_str());
  //The || operator
  printf("(%d == %d) && (%d > %d) is %s.\n", a, c, b, d, TorF[(a == c) || (b == d)].c_str());
  //The 'Not' operator
  printf("!(%d == %d) is %s.\n", a, c, TorF[!(a == c)].c_str());

  return 0;
}
```



```
(10 == 10) && (5 > 3) is True.
(10 == 10) && (5 > 3) is True.
!(10 == 10) is False.
Press any key to continue . . .
```

Conditional execution

- Boolean expression: evaluate to true/false
  - What is true? What is false?
  - bool type
  - Type conversion
  - Assignment
  - Common expressions

Conditional execution

- Type bool: true / false
  - Size: 1 byte (basic unit of storage)
  - Be represented as integer: true = 1, false = 0
- What happens when you assign a value to boolean type:
  - False: 0 value (for integer, floating point number, character '\0')
  - True: anything else (except structures, unless a casting operator is defined)

Conditional execution

- Examples:

  -
    ```
    bool b = true, b1 = false;
    int a = -1, c = 0;
    float x = 0.5f, y = 1.2f;
    b = a > c;
    b1 = a;
    b = c;
    b1 = x < y && a > c;
    b = x;
    c = y + b1;
    b1 = 50 != ‘a’;
    b = x + 4.9 < y / 0.5f;
    ```

If-else statement

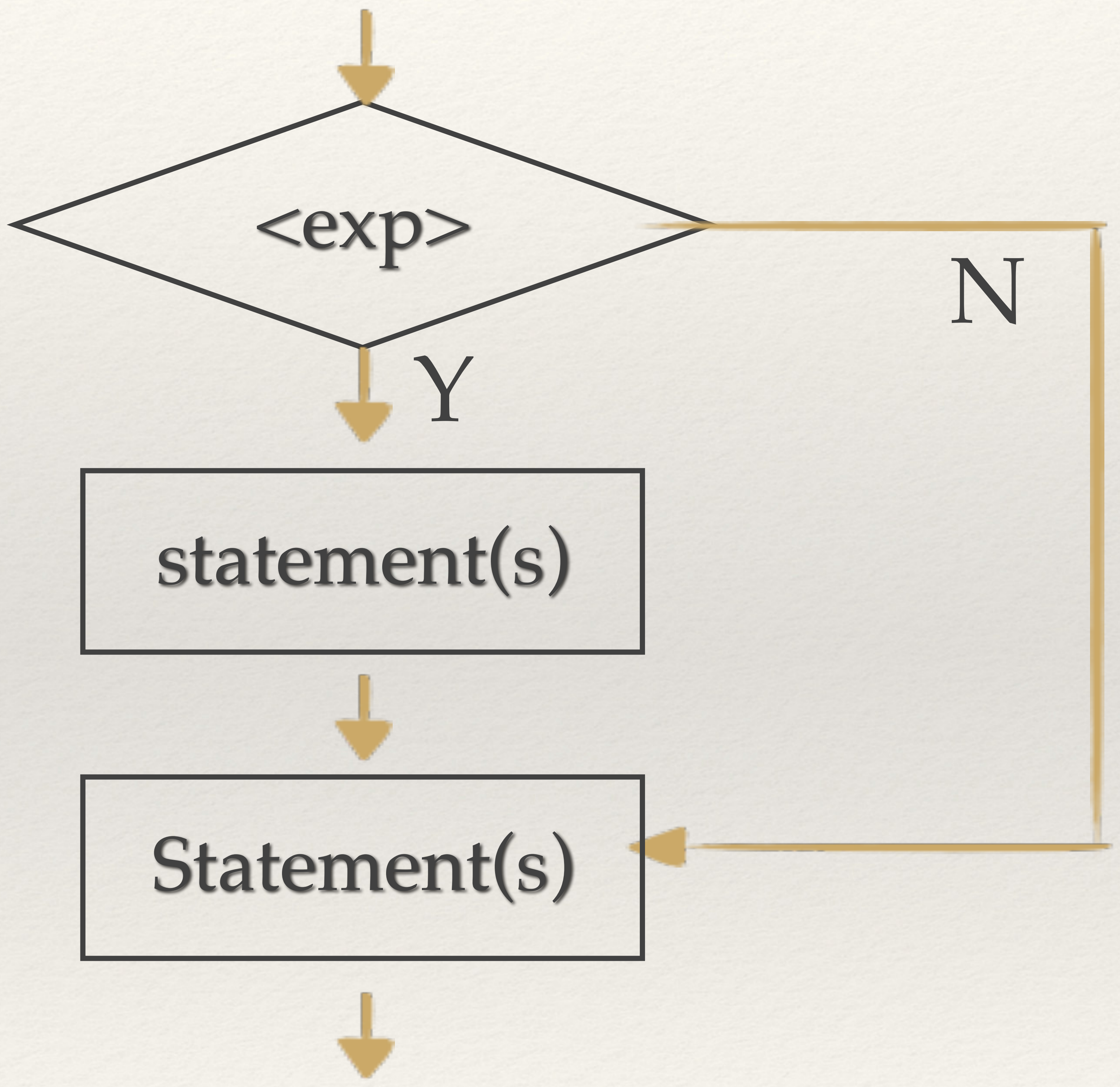
If statement

- Simple if statement:
  - Execute a statement or a list of statements if the given condition is satisfied

  -
    ```
    if (<conditional expression>) <statement>;
    ```

  -
    ```
    if (<conditional expression>) {
        <statements>
    }
    ```

Flowchart



Example

```
#include <iostream>
using namespace std;

int main() {
   int number;
   cout << "Enter an integer: ";
   cin >> number;

   if (number > 0)
       cout << "You entered a positive integer: " << number << endl;

   cout << "This statement is always executed.";
   return 0;
}
```

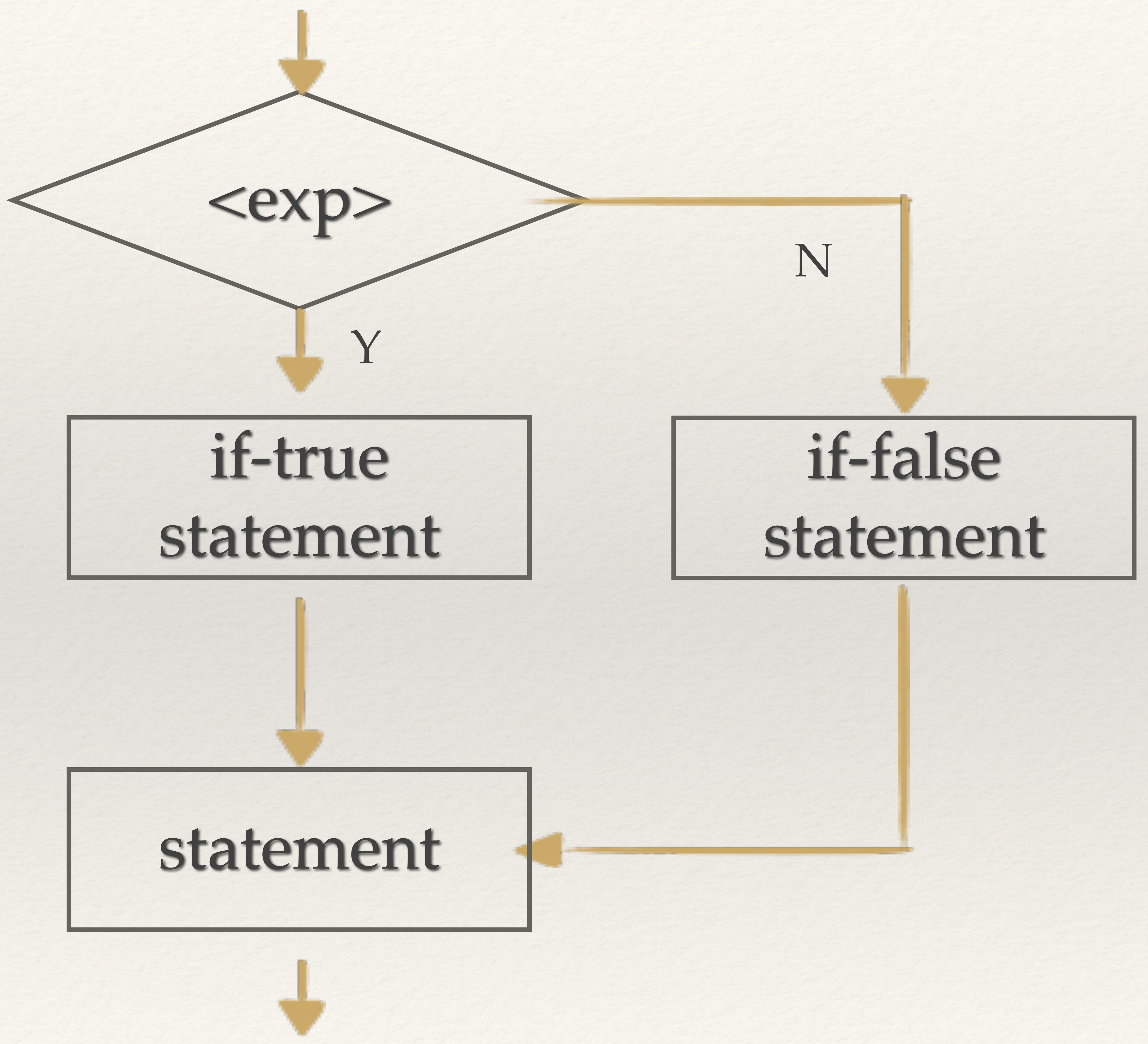
If-else statement

- Full if-else statement:

  -
    ```
    if (<conditional expression>) <if-true statement>;
    else <if-false statement>;
    ```

  -
    ```
    if (<conditional expression>) {
        <if-true statements>
    }
    else {
        <if-false statements>
    }
    ```

Flowchart



Example

```
#include<iostream>
using namespace std;

int main() {
   int a, b, max;
   cout << "Enter two integer numbers: ";
   cin >> a >> b;
   if (a > b)
       max = a;
   else
       max = b;

   printf("The maximum value between %d and %d is %d\n", a, b, max);
   return 0;
}
```

Nested conditionals

- Nested if-else statements

  -
    ```
    if (<exp>)                          // first check
         if (<exp>)                     // second check
             if (<exp>)                 // third check
                 <statement>
             else <statement>
         else <statement>
    else if (<exp>) <statement>
         else if (<exp>) <statement>
              else <statement>
    ```

Nested conditionals

- Nested if-else statements: multi-way

  -
    ```
    if (<exp 1>) <statement 1>
    else if (<exp 2>) <statement 2>
    else if (<exp 3>) <statement 3>
    else <statement 4>
    ```

  -
    ```
    if (<exp 1>) <statement 1>
    else if (<exp 2>) <statement 2>
         else if (<exp 3>) <statement 3>
              else <statement 4>
    ```

Example

```cpp
#include<iostream>
using namespace std;
int main() {
   float score;
   char grade;
   cout << "Enter a score [0 - 10]: ";
   cin >> score;
   if (score >= 8)
       grade = 'A';
   else if (score >= 6.5)
       grade = 'B';
   else if (score >= 5)
       grade = 'C';
   else if (score >= 4)
       grade = 'D';
   else
       grade = 'F';

   cout << "Your grade is " << grade << endl;
   return 0;
}
```

Conditional operator

- Syntax:
  - `<expression> ? <if-true expression> : <if-false expression>`
  - Equivalent to if-else statement but apply for expressions
- Example:

  -
    ```
    char outChar;
    outChar = a == ‘c’ ? ‘C’ : ‘c’;
    ```

  -
    ```
    float diff;
    diff = x > y ? x - y : y - x;
    ```

Example

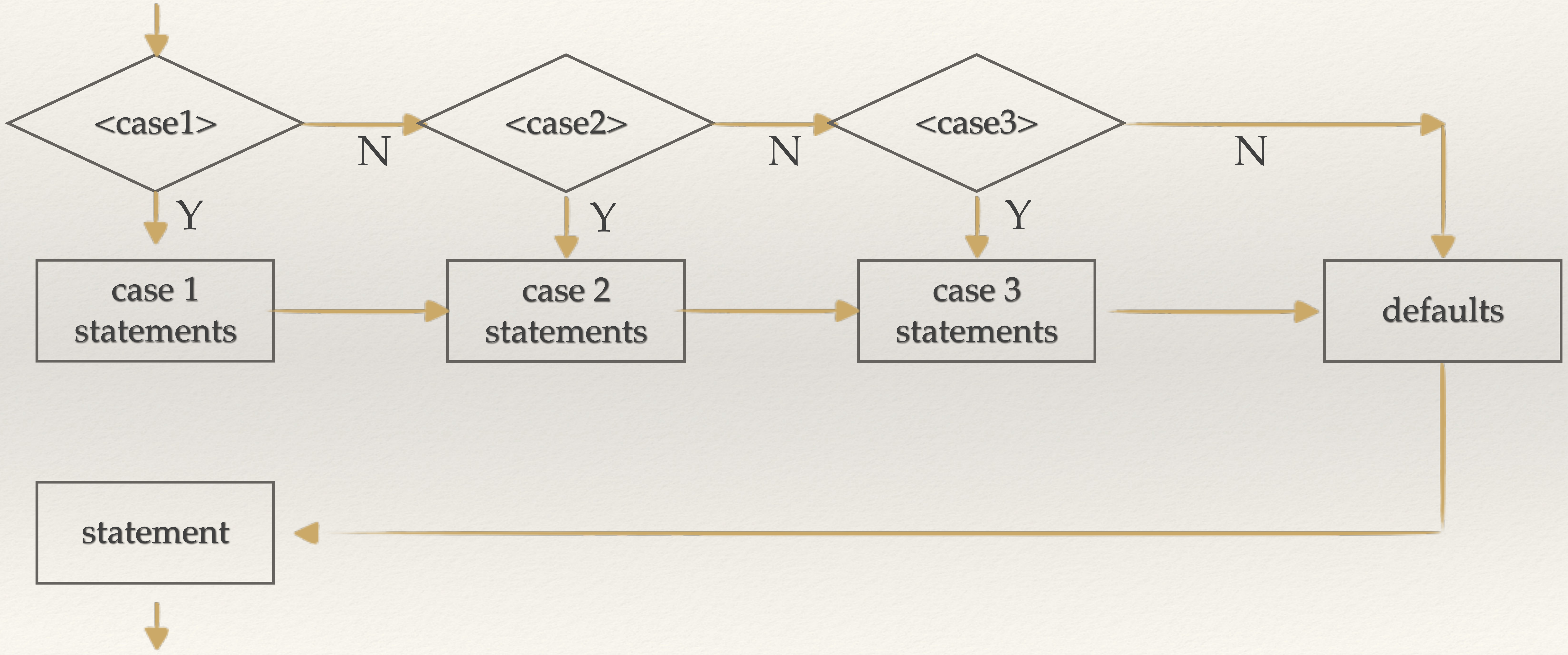
```
#include<iostream>
using namespace std;

int main()
{
   int a, b, max;
   cout << "Enter two integer numbers: ";
   cin >> a >> b;
   max = (a > b) ? a : b;

   printf("The maximum value between %d and %d is %d\n", a, b, max);
   return 0;
}
```

Switch statement

Switch statement

- A convenient way to write multi-way statement
-
```
switch(<exp>) {
    case <value 1>: <statements>; break;
    case <value 2>: <statements>; break;
    …
    case <value N>: <statements>; break;
    default: <statements>;
}
```

Flowchart


<case1>
<case2>
<case3>
N
N
N
Y
Y
Y
case 1 statements
case 2 statements
case 3 statements
defaults
statement

Example

```
#include <iostream>
using namespace std;

int main() {
   char o;
   float num1, num2;

   cout << "Enter an operator (+, -, *, /): ";
   cin >> o;
   cout << "Enter two operands: ";
   cin >> num1 >> num2;

   switch (o) {
   case '+':
      cout << num1 << " + " << num2 << " = " << num1 + num2 << endl;
      break;
   case '-':
      cout << num1 << " - " << num2 << " = " << num1 - num2 << endl;
      break;
   case '*':
      cout << num1 << " * " << num2 << " = " << num1 * num2 << endl;
      break;
   case '/':
      cout << num1 << " / " << num2 << " = " << num1 / num2 << endl;
      break;
   default:
      cout << "Error! operator is not correct";
      break;
  }

return 0;
}
```

What happens if no break statement?

-
  ```
  switch(<exp>) {
        case <value 1>: <statements>;
        case <value 2>: <statements>;
        …
        case <value N>: <statements>;
        default: <statements>;
  }
  ```

If break statement is not used, all cases after the correct case is executed.

Enumerated type

- Define a list of possible values of a type
  - `enum <type name> {<name of possible values>};`
  - `enum [<type name>] {<name of possible values>} <variables>;`
- Example:

  -
    ```
    enum Color {Red, Orange, Yellow, Green, Blue, Violet};
    Color c = Yellow;
    cout << “Yellow color has value: ” << c << endl;
    ```

Enumerated type

- Define a list of possible values of a type
  - `enum <type name> {<name0 = value0>, <name1 = value1>, …};`
  - `enum [<type name>] {<name0>} <variables>;`
- Example:

  -
    ```
    enum Color {Red = -1, Orange = 2, Yellow = 8, Green = 3, Blue,
    Violet};
    Color c = Blue;
    cout << "Blue color has value: " << c << endl;
    ```

Why enums are used in C++ programming?

```
#include <iostream>
using namespace std;

enum suit {club = 0, diamonds = 10, hearts = 20, spades = 3} card;

int main()
{
   card = club;
   cout << "Size of enum variable " << sizeof(card) << " bytes.\n";
   return 0;
}
```

How to use enums for flags?

```cpp
#include <iostream>
using namespace std;

enum designFlags {
BOLD = 1,
ITALICS = 2,
UNDERLINE = 4
};

int main()
{
   int myDesign = BOLD | UNDERLINE;
   cout << myDesign;
   return 0;
}
```

Preprocessor directives

Preprocessor directives

- Conditional Pre-processor directives:
  - `#define, #undef, #ifdef, #ifndef, #else, #elif, #endif`
  - E.g.:

    -
      ```
      int foo(float a, double b) {
      #ifdef __MSC_VER
          return a * 3.14159 - sqrt(b * a);
      #else
          return a * 3.14159 + b * b;
      #endif
      }
      ```

Preprocessor directives

- Conditional Pre-processor directives
  - Library headers (*.h, *.hpp):

    -
      ```
      #pragma once
      // library definition
      ```

    -
      ```
      #ifndef __MY_LIBRARY_H__
      #define __MY_LIBRARY_H__
      // library definition
      #endif
      ```

Preprocessor directives

- Power of macros and preprocessor directive
  - One definition fit all
  - Flexible, portable
  - Open source community

Indents, coding style

- Use indents to enhance your code
  - Easy to manage flow of code
  - Easy to read code
- Coding requires skills and the programmer, in most of cases need to follow rules of their community.

Summarise

- Understand basic elements of C/C++
  - Principle of conditional execution
  - if-else statement, nested conditionals
  - switch statement
  - Conditional operator