bool [C++]

- `bool` is short for “boolean”, named after George Boole.
- In C++ the built-in data type `bool` can have the value `false` or `true`, e.g.

```
bool flag = true;
```

- `bool` is the simplest data type.
- Also, an `int` can either be `false` (zero) or `true` (non-zero), and used as a boolean value. The C language does this instead of having a `bool` data type. So C has no `bool` data type.
- Boolean expressions are formed by applying relational (and equality) operators to arithmetic expressions, e.g.

```
bool flag = 2 + 3 < 20%5;
```

- or applying `&&`, `||`, `!` to `bool` expressions, much like with an `if` statement.

if, else if, else.

- Conditions in C++ are based on Boolean values, and have only two possible values, `true` or `false`. We use `if` statements to test conditions.
- Probably the most important statement in C++ if the `if` statement.
- `if` a certain *condition* is true then the *control statement* is run.

```
if (condition)
{
      // control statement
}
```

- here, *condition* is the condition being tested, and the *control statement* is the code that is run if the condition is true.
- Note that the *condition* is in parentheses to separate it from the rest of the *control statement*.

```
if (5 < 10)
{
      cout << "Five is always less than ten." << endl;
}
```

- The simplest condition to test for is just the Boolean `true`.

```
if (true)
{
      cout << "true is always true" << endl;
}
```

- Typically the condition tests a variable, e.g.,

```
int tempFahrenheit = 20;
if (tempFahrenheit < 32)
{
    cout << "Water freezes at 32 F." << endl;
}
```

- Sometimes you might want to take an alternative action if a condition does not hold.
- Use the `else` statement to create an alternate action, e.g.,

```
int cats = 2;
if (cats < 4)
{
```

```
        cout << "She has " << cats << " cats." << endl;
    }
    else
    {
        cout << "She is crazy, and has " << cats << " cats." << endl;
    }
```

- An `else` statement cannot occur by itself and must proceed an `if`.
- You may have include one or more `else if` conditions, which is often referred to as an "if then else block." e.g.,

```
    int cats = 2;
    if (cats < 3)
    {
        cout << "She only has " << cats << " cats." << endl;
    }
    else if (cats < 5)
    {
        cout << "She has " << cats << " cats." << endl;
    }
    else if (cats < 7)
    {
        cout << "She is crazy, and has " << cats << " cats." << endl;
    }
    else
    {
        cout << "She is completely crazy, and has " << cats << "
cats." << endl;
    }
```

- The simplest Boolean operator is the equivalence operator ==, used to test values.
- Do not confuse the equivalence operator (==) with the assignment operator, which is one equal sign (=). The assignment operator assigns the value while the equivalence operator tests the value.

```
    int cats = 2;
    if (cats == 1)
    {
        cout << "She has one cat." << endl;
    }
    else if (cats < 4)
    {
        cout << "She has " << cats << " cats." << endl;
    }
    else
    {
        cout << "She is crazy, and has " << cats << " cats." << endl;
    }
```

The Conditional Operator (AKA Ternary Condition)

- Some conditions are trivial, making it tedious to write a complete if statement for them.

```
if (x > y)
{
    z = y;
}
else
{
    z = x;
}
```

- C++ provides a conditional operator as shorthand for such trivial if conditions.
- The above if block in C++ may be expressed as,

```
z = (x > y) ? y : x;
```

- This is also known as a ternary condition. "ternary" as it is composed of three parts.

Conditional Operators and Compound Conditions

==	Equal. Note TWO equal signs. Don't use one, it will try to assign.
!=	Not equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
&&	and
||	or
!	not

Example Checking Upper and Lowercase

- You can, for instance, use the || (or) condition two check for upper or Lowercase, e.g.,

```
char ch;
cout << "Choose a letter: ";
cin >> ch;

if (ch == 'Z' || ch == 'z')
{
     cout << "You chose wisely." << endl;
}
```

- You can use the ASCII codes to determine if the input is upper or Lowercase, e.g.,

```
if (ch >= 65 && ch <= 90)
{
     cout << "You chose Uppercase." << endl;
```

```
}
else if (ch >= 97 && ch <= 122)
{
      cout << "You chose Lowercase." << endl;
}
else
{
      cout << "You chose poorly." << endl;
}
```

- You can even use the ASCI conversions in the if statement, e.g.,

```
if (ch == 'Z' || ch == 'z')
{
    cout << "You chose wisely." << endl;
}

if (ch >= 'A' && ch <= 'Z')
{
    cout << "You chose Uppercase." << endl;
}
else if (ch >= 'a' && ch <= 'z')
{
    cout << "You chose Lowercase." << endl;
}
else
{
    cout << "You chose poorly." << endl;
}
```

- You can use `char` conversion to `int` to convert Lowercase to Uppercase.

```
if (ch > 'Z')
{
      // convert Lowercase to Uppercase
      int asciiDifference = ('a' - 'A');
      ch = ch - asciiDifference;
}

cout << ch << endl;
```

- Or Uppercase to Lowercase

```
if (ch < 'a')
{
      // convert Uppercase to Lowercase
      int asciiDifference = ('a' - 'A');
      ch = ch + asciiDifference;
```

```
    }

    cout << ch << endl;
```

- It's important understand what the two above examples do, but in practice you should simply use `ch = toupper(ch);` or `ch = tolower(ch);` when converting case on a `char`, more on those later.

Switch Blocks

The second type of selection statement is the `switch` statement.
Switch blocks are most commonly used when a programmer wants to evaluate a user's response to a specific set of choices. e.g.,

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

int main()
{
    cout << "Type a letter: " << endl;

    char ch;
    cin >> ch;

    switch (ch)
    {
        case 'a':
            cout << "you typed a" << endl;
            break;
        case 'b':
            cout << "you typed b" << endl;
            break;
        case 'c':
            cout << "you typed c" << endl;
            break;
        case 'd':
        case 'e':
        case 'f':
            cout << "you typed d, e, or f" << endl;
            break;
        default:
            cout << "Wrong answer." << endl;
    }

    return 0;
}
```

Switch blocks are very powerful and easy to read but it's easy to forget the `break` statement.