Introduction to Programming (CS 101)

Spring 2024

Lecture 3:

Conditional (if, if-else, switch) blocks, Boolean expressions
ternary operator

Instructor: Preethi Jyothi

Recap

What is the output from the following piece of code?

0

2

4

1

```
#include <simplecpp>

main_program{

  int i = 1, j = 2, k = 5;
  k += i - 2 * j - 4 / 2;
  cout << k << endl;

}
```

Homework problems

[Q1]. Write C++ code to calculate the following series: $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \ldots$

`alt-harmonic.cpp`

[Q2]. Write C++ code to calculate $\sin(x)$ using the Taylor series expansion (where x is in radians):

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Ask for x (in degrees) from the user and use a fixed number of terms. You can use `PI` (offered by `simplecpp`) to access the value of π.

`sine.cpp`

Another useful arithmetic operator: % (modulus)

- `%`: the modulo or modulus (remainder from division) operator

  `operand1` `%` *`operand2`*:

  Works on integers and finds the remainder of *`operand1`* `/` *`operand2`*

- Has the same level of precedence as multiplication and division
- Left-to-right associativity, i.e. `a % b % c` will be processed as `((a % b) % c)`

`if` statement

CS 101, 2025

`if` statement

- Syntax:

```
if(condition) {
    body
}
```

- Semantics: **body** should be executed if *condition* is true

- *condition* is an expression that takes the value "false" (zero) or "true" (non-zero)

- *condition* can include relational operators.
  Examples: `i <= j, i > j, i >= j, i < j, i == j, i != j`

Logical operators

```
if(condition) {
    body
}
```

- *condition* can also include logical operators, that return the result of a Boolean operation

1. Logical NOT denoted by '!'

   !*expression* evaluates to true if *expression* evaluates to false, and vice-versa

2. Logical AND denoted by '&&'

   operand1 && *operand2* evaluates to true if both *operand1* and *operand2* are true
   Otherwise, the result is false.

3. Logical OR denoted by '||'

   operand1 || *operand2* evaluates to true if either *operand1* or *operand2* or both are true

Boolean variables

- Like integers takes values such as ..., -2, -1, 0, 1, 2, ..., Boolean variables take values `true`, `false`

- Arithmetic operators such as `+`, `*`, etc. are used with integers, floats; Boolean variables allow operations like && (AND), `||` (OR), `!` (NOT)

- Data type `bool` in C++ is used to represent Boolean variables. Example:
  `bool done = false;` // equivalent to `bool done = 0;`

Boolean expressions

- Consider two Boolean variables `x` and `y`
  - `!x` → NOT `x` is true if and only if `x` is false
  - `x && y` → `x` AND `y` is true if and only if both `x` and `y` are true
  - `x || y` → `x` OR `y` is true if and only if at least one of `x` or `y` is true
- A variable of type `bool` can be used to save the value of a Boolean expression

```
bool choice;
choice = (input == 'S' || input == 'D');

if(!choice)
  cout << "Invalid input. Try again.";
```

Evaluating Boolean expressions

- To evaluate *`condition1`* `||` *`condition2`* first *`condition1`* is evaluated
  - if *`condition1`* is true, then the entire expression is taken to be true without evaluating
  - otherwise, *`condition2`* is evaluated and its value becomes the value of the expression

- Similarly, while evaluating *`condition1`* `&&` *`condition2`* if *`condition1`* is false, the entire expression is taken to be false, without evaluating *`condition2`*

- Operator precedence: `!` is evaluated first, followed by `&&` and then `||` [1]

```
bool out = !a && b || c;
```

Here, `out` is `((!a) && b) || c`

[1] https://en.cppreference.com/w/cpp/language/operator_precedence

Code to draw a square or diamond depending on user choice

Demo in class and code shared on Moodle

Code with `if` statement

```
cout << "Do you want a square or diamond? Use S or D";
char input;
cin >> input;

if(input == 'D') {
  right(45);
}

repeat(4) {
  forward(100); right(90);
}
```

if the body of `if` is a single statement,
`{` and `}` can be omitted

Code with `if` and logical operators

```
cout << "Do you want a square or diamond? Use S or D";
char input;
cin >> input;

if(input == 'D')
  right(45);

if(input == 'D' || input == 'S') {
  repeat(4) {
    forward(100); right(90);
  }
}

if(input != 'D' && input != 'S')
  cout << "Input should be either D or S. Try again.\n"
```

Can omit `{}` since repeat is a single statement

Logical operators `||` and `&&`

Recall `\n` is a newline character

Code with `if` and logical operators

```
cout << "Do you want a square or diamond? Use S or D";
char input;
cin >> input;

if(input == 'D')
  right(45);

if(input == 'D' || input == 'S')
  repeat(4) {
    forward(100); right(90);
  }


if(input != 'D' && input != 'S')
  cout << "Input should be either D or S. Try again.\n"
```

if-else block

```
cout << "Do you want a square or diamond? Use S or D";
char input;
cin >> input;

if(input == 'D')
  right(45);

if(input == 'D' || input == 'S')
  repeat(4) {
    forward(100); right(90);
  }
else {
  cout << "Input should be either D or S. Try again.\n";
}
```

Other forms of `if` statement

CS 101, 2025

`if-else` statement

- Syntax:

```
if(condition){
    body1
} else {
    body2
}
```

- Semantics: run **body1** if *condition* is true and run **body2** if *condition* is false

- Equivalent to:

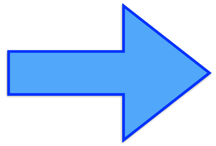
```
if(condition){ body1 }

if(!condition){ body2 }
```

Chaining `if-else` statements

- Syntax:

```
if(condition1) {
    body1
} else if(condition2) {
    body2
} else if(condition3) {
    body3
} else {
    body4
}
```



```
if(condition1) {
    body1
} else {
    if(condition2) {
        body2
    }
    else {
        if(condition3) {
            body3
        } else {
            body4
        }
    }
}
```

- Semantics: run **body1** if *condition1* is true, run **body2** if *condition2* is true, run **body3** if *condition3* is true. If none of these conditions match, then run **body4**.

switch operator

CS 101, 2025

`switch` statement

- Transfers control to one of several code blocks, depending on the value of *`condition`*

```
switch(condition) {
  case constant1:
  body1
  break;

  case constant2:
  body2
  break;

  default:
  // execute if none of
  // the constants are matched
  body3
}
```

- Semantics: *`condition`* is evaluated and compared with the values of each case value (*`constant1`*, *`constant2`*). Run the corresponding code (**body1**, **body2**) if there is a match.

- `break` inside each case block terminates the switch statement. If no break, all the statements after the matching case will be executed until a `break` is encountered.

- If there is no match, then run **body3** after `default`.

Some rules of `switch`

- Case values (*`constant1`*, *`constant2`*) must be either `int` or `char` type
- No duplicate case values are allowed
- `default` is an optional block
- `break` statements inside each case block are optional
- A case block can be empty
- Switch conditions can be expressions that evaluate to an `int` or `char` value. Example: `switch(1+2-3)` is allowed.
- Position of the default case does not matter; can appear in the beginning, middle or end. Typical to include it in the end.

ternary operator

CS 101, 2025

Ternary conditional operator

- What does this program do?

```
#include <simplecpp>
main_program {
    int a, b, c;
    cin >> a >> b;
    if(a > b)
      c = a;
    else
      c = b;
}
```

equivalent code using a *ternary operator*

```
#include <simplecpp>
main_program {
    int a, b, c;
    cin >> a >> b;
    c = ((a > b) ? a : b);
}
```

- Semantics of *condition1* ? *statement1* : *statement2*: if *condition1* evaluates to true, then run *statement1*; otherwise, run *statement2*

Next class: Looping Statements

CS 101, 2025