Introduction to control flow

William Hendrix,

presented by:

Nan Jiang

Lecture 3

Outline

- Basic I/O
- Boolean expressions
- Basic control flow
- Program design

Printing values to the screen

- `cout` : "console output"

- Syntax: `cout << [thing to output];`
  - Mnemonic: values go out to the screen to be printed
  - Use double quotes for messages
  - Numerical values or variables don't need quotes
  - Does *not* start a new line automatically
    - `cout << endl;`
    - `cout << '\n';`
    - `cout << "Message\n";`
- Shortcut: `cout << [1st thing] << [2nd thing...];`

Getting values from the user

- `cin`: "console input"

- Syntax: `cin >> [variable];`
  - Mnemonic: values come from the console and go into variables
- Prompts user but does not print anything
- Pauses program until user presses enter
- You should always check input from user (later)
- Only reads first "word" entered by user
- Anything not used for assignment stays in `cin`
  - Includes errors like entering letters for numerical variables
  - User will not be prompted again until all input is cleared
  - To clear unused input: `cin.sync();`

Boolean operators

- Six comparison operators
  - Compare two values and produce a Boolean value
  - ==, `!=` (not equals), <, <=, >, >=
  - E.g., `1 < 0` (false), `2.0 == 2` (true)
  - Pitfall: DO NOT CONFUSE == WITH =
- Three connective operators
  - Logical not (`!`)
  - Logical and (`&&`)
  - Logical or (`||`) <- One or the other *or both*
  - DeMorgan's Laws
    - `!(x && y)` -> `!x || !y`
    - `!(x || y)` -> `!x && !y`
- Pitfall: `0 <= 200 <= 100` (true)
  - Use `0 <= 200 && 200 <= 100`, instead

Basic control flow in C++

- Syntax

```
if (condition) // <- No semicolon!!
    <statement>;
if (condition) // Open brace can go here
{
  //Block executed if condition is true
}
```

- `condition` is any Boolean variable or expression
  - Holdover from C: any non-zero value is considered true
- **Good style**: always indent the statement(s) after an `if`
- C/C++ is whitespace insensitive
  - Extra spaces, new lines, and tabs are ignored
  - Use these to make code more readable

Dealing with bad input (the easy way)

- We should always check input from user
  - Users might mistype, ignore directions, or be malicious

```
cout << "Enter a positive number: ";
cin >> number;
if (number <= 0)
{
  cout << "That is not a positive number\n";
  return 1; //Exits program with error code 1
}
```

- We'll learn a better way to handle this when we discuss loops

Alternate flows

- When `condition` is false, we can use `else` and else if to take another branches

- Syntax

```
if (condition1)
  <statement/block if condition1 true>
else if (condition2)
  <statement/block if condition2 true>
//else if #3...
else
  <statement/block if nothing else true>
```

Nested conditions

- You can nest `if` statements inside others

```
if (hasNoFactors)
{
  if (number > 1)
    cout << "prime";
}
else
  cout << "composite";
```

- Warning: the dangling else
  - `else` and `else if` associate with nearest `if` statement
  - Without braces, `else` would "belong" to 2nd `if`
    - Would print composite if `hasNoFactors` and `number <= 1`

Alternative to multiple `if ... else if`

```
switch (variable)
{
  case 1:
    <1+ stmts executed if variable == 1>
    break;
  case 42:
     //...
     break;
  default:
     <executed if nothing else is true>
}
```

- `variable` must be an integer type
- Slightly faster than `if ... else if ... else`
- Following cases will execute if you forget `break`

Program design

- Program of any complexity should be planned out before starting to code
- Two main strategies: top-down and bottom-up

- Top-down design (recommended)
  - What are the steps needed to solve this problem?
  - What are the different cases (`if` statements)?
  - What variables do I need to keep track of?
    - What types are they?

- Bottom-up design (case-based reasoning)
  - What is a typical problem instance?
    - How would I solve this?
  - What are the special cases, and how does our algorithm change?

Design exercise

- Write a problem that prompts the user for an integer and outputs whether the integer is odd or even

Sample solution:

```
int num;
cout << "Enter a number: ";
cin >> num;
if (cin.fail())
{
  cout << "Input error\n";
  return 1;
}
if (num % 2 == 0)
  cout << num << " is even\n";
else
 cout << num << " is odd\n";
//Optional:  prompt user again so window doesn't close
```

Tonight

Recommended reading: Sections 3.6-3.8, 3.10, Top-down design handout