Solving Problems
Flow Control in C++

CS 16: Solving Problems with Computers I
Lecture #3

Ziad Matni
Dept. of Computer Science, UCSB
A Word About Registration for CS16

FOR THOSE OF YOU NOT YET REGISTERED:

• There’s still a waitlist to add this class!
  – We now have a few openings and I will go by the prioritized waitlist

→ WAITLISTED STUDENTS MUST SEE ME AFTER CLASS ←
Lecture Outline

• Problem Solving

• Simple Flow of Control
• IF/ELSE Statements
• Loops (While ; Do-While ; For)
• Multiway Branching and the switch command
• Local vs. Global Variables

• Some Notes on Program Style and Errors
How Does One Solve Problems?

Understand the problem

Devise a plan

Carry out the plan

Look back and re-assess
Strategies

**Ask questions!**

- *What do I know about the problem?*
- *What is the information that I have to process in order to find the solution?*
- *What does the solution look like?*
- *What sort of special cases exist?*
- *How will I recognize that I have found the solution?*
Strategies

Ask questions! Don’t reinvent the wheel!

Similar problems come up again and again in different guises

A good programmer recognizes a task that has been solved before and can research the solution

However, a good programmer does not plagiarize...
Divide and Conquer!

**Break up** a large problem into smaller units and solve each smaller problem.

Applies the concept of abstraction.

The divide-and-conquer approach can be applied over and over again until each subtask is manageable.
Computer Problem-Solving

Analysis and Specification Phase
- Analyze the problem
- Specify the details

Algorithm Development Phase
- Develop an algorithm
- Test your algorithm

Implementation Phase
- Code your algorithm
- Test your code

Maintenance Phase
- Use the program
- Maintain the program

Can you see a recurring theme?
Developing Software Products

• As a business product
  – Software is “made” (developed) to meet market needs

• Needs resources and planning
  – Software needs to be
    programmed, documented, tested, fixed/maintained

• There is a process to everything you need to do!
  – A complex task – a problem to solve – needs a plan, an algorithm
A structured approach to software development:

**GOAL:** A software development process that leads to:

- a high quality system that meets or exceeds customer expectations,
- within time and cost estimates,
- works **effectively** and **efficiently** in the current and planned infrastructure,
- and is **cheap** to maintain and **cost effective** to enhance.
Software Systems Development: Waterfall Model

- Requirement Analysis (Both Business & Tech.)
- Systems Design
- Implementation (Programming)
- Testing
- Deployment & Maintenance
Flow of Control

• Another way to say: *The order in which statements get executed*

• Branch: *(verb)* How a program chooses between 2 alternatives
  – Usual way is by using an *if-else* statement

```
if (Boolean expression)
    true statement
else
    false statement
```
Implementing IF/ELSE Statements in C++

• As simple as:

```cpp
if (income > 30000) {
    taxes_owed = 0.30 * 30000;
}
else {
    taxes_owed = 0.20 * 30000;
}
```

Where's the semicolon??!

Curly braces are optional if they contain only 1 statement.
**IF/ELSE in C++**

- To do additional things in a branch, use the `{ }` brackets to keep all the statements together.

```cpp
if (income > 30000)
{
    taxes_owed = 0.30 * 30000;
    category = "RICH";
    alert_irs = true;
} // end IF part of the statement
else
{
    taxes_owed = 0.20 * 30000;
    category = "POOR";
    alert_irs = false;
} // end ELSE part of the statement
```

Groups of statements (sometimes called a **block**) kept together with `{ ... }`
Examples of IF Statements

```java
if ( (x >= 3) && (x < 6) )
y = 10;
```

- The variable y will be assigned 10 only if x is equal to 3, 4, or 5

```java
if !(x > 5) y = 10;
```

- The variable y will be assigned 10 if x is NOT larger than 5 (i.e. if x is 4 or smaller)
  - DESIGN PRO-TIP: Unless you really have to, avoid the NOT logic operator when designing conditional statements
Beware: = vs ==

• '=' is the **assignment** operator
  – Used to assign values to variables
  – Example: \texttt{x = 3;}

• '==' is the **equality** operator
  – Used to compare values
  – Example: \texttt{if ( x == 3) y = 0;}

• The compiler will actually accept this logical error: \texttt{if (x = 3) y = 0;}
  – **Why?**
  – It’s an error of logic, not of syntax
  – But it stores 3 in \texttt{x} instead of comparing \texttt{x} and 3
  – Since the result is 3 (non-zero), the expression is true, so \texttt{y} becomes 0
Simple Loops 1: \textit{while}

- We use loops when an action must be repeated
- C++ includes several ways to create loops
  - while, for, do...while, etc...

- The \textbf{while loop} example:

```cpp
int count_down = 3;
while (count_down > 0)
{
    cout << "Hello ";
    count_down -= 1;
}
```

Output is: \texttt{Hello Hello Hello}

Where's the semicolon??!
Simple Loops 2: **do-while**

- Executes a block of code *at least once*, and then repeatedly executes the block depending on a given Boolean condition at the end of the block.
  - So, unlike the while loop, the Boolean expression is checked *after* the statements have been executed.

```cpp
int flag = 1;
do{
    cout << "Hello ";
    flag -= 1;
}while (flag > 0);
```

Output is: **Hello**

Why is there a semicolon here?!?
Simple Loops 3: *for*

- Similar to a while loop, but presents parameters differently.
- Allows you to initiate a *counting variable*, a *check condition*, and a way to *increment your counter* all in one line.

```
for (counter declaration; check condition statement; increment rule) {...}
```

Example:
```
for (int count = 2; count < 5; count++)
{
    cout << "Hello ";
}
```

Output is:
```
Hello Hello Hello
```
Increments and Decrements by 1

In C++ you can increment-by-1 like this:

more common → \( a++ \)

or like this:

\( ++a \)

Similarly, you can decrement by:

\( a-- \) or \( --a \)
In a while loop, you always need to increment a counter var.

Example:

```cpp
int max = 0;
while (max < 4)
{
    cout << "hi" << endl;
    max++;
}
```

What will this print out?
Some Cool Uses of \texttt{x++}

• You can make a slight change and save a line of code!

Example:

```cpp
int max = 0;
while (max++ < 4)
{
    cout << "hi" << endl;
}
```
When to use `x++` vs `++x`

- `x++` will assess `x` *then* increment it
- `++x` will increment `x` *first, then* assess it

- 95% of the time, you will use the first one

- In `while` statements, it *makes* a difference
- In `for` statements, it *won’t make* a difference
Examples

for (int c = 0; c < 4; c++)
    cout << “hi” << endl;

Prints “hi” 4 times

for (int c = 0; c < 4; ++c)
    cout << “hi” << endl;

Prints “hi” 4 times

int max = 0;
while (max++ < 4)
{
    cout << “hi” << endl;
}

Prints “hi” 3 times

int max = 0;
while (++max < 4)
{
    cout << “hi” << endl;
}

Prints “hi” 4 times
Infinite Loops

• Loops that never stop – to be avoided!
  – Your program will either “hang” or just keep spewing outputs for ever

• The loop body should contain a line that will eventually cause the Boolean expression to become false (to make the loop to end)

• Example: Goal: Print all positive odd numbers less than 6

  ```cpp
  x = 1;
  while (x != 6)
  {
    cout << x << endl;
    x = x + 2;
  }
  ```

What is the problem with this code?

What simple fix can undo this bad design?

```cpp
while (x < 6)
```
Using **for-loops** For Sums

- A common task is reading a list of numbers and computing the sum
  - Pseudocode for this task might be:
    ```
    sum = 0;
    repeat the following this_many times
      get input for "next"
      sum = sum + next
    end of loop
    ```

- Let’s look at it as a for-loop in C++ ...
Using **for-loops** For Sums

- The pseudocode from the previous slide can be implemented as

  ```
  int sum = 0;
  for(int count = 0; count < 10; count++)
  {
    cin >> next;
    sum = sum + next;
  }
  ```

- Note that “sum” must be initialized prior to the loop body!
  - Why?
Using *for-loops* For Products

- Forming a **product** is very similar to the sum example seen earlier

```c
int product = 1;
for(int count = 0; count < 10; count++)
{
    cin >> next;
    product = product * next;
}
```

- Note that “product” must be initialized prior to the loop body
  - Product is initialized to 1, *not* 0!
Ending a While Loop

• A for-loop is generally the choice when there is a predetermined number of iterations
• When you DON’T have a predetermined number of iterations, you will want to use while loops

The are 3 common methods to END a while loop:

• List ended with a sentinel value: Using a particular value or calculation to signal the end
• Ask before iterating: Ask if the user wants to continue before each iteration
• Running out of input: Using the eof function to indicate the end of a file (more on this when we discuss file I/Os)
List Ended With a Sentinel Value

cout << "Enter a list of positive integers.\n" << "Place a negative integer after the list to quit.\n";
sum = 0;
cin >> number;
while (number > 0)
{
    cout << “The double of that is: ” << 2*number << endl;
    cin >> number;
}

– Notice that the sentinel value is read, but not processed at the end
Ask Before Iterating

```cpp
sum = 0;
char ans;

cout << "Are there numbers in the list (Y/N)?"; cin >> ans;

while ((ans == 'Y') || (ans == 'y'))
{
    // statements to read and process the number
    cout << "Are there more numbers(Y/N)? "; cin >> ans;
}
```
Nested Loops

• The body of a loop may contain any kind of statement, including another loop.

• When loops are nested, all iterations of the inner loop are executed for each iteration of the outer loop.

• ProTip: Give serious consideration to making the inner loop a function call to make it easier to read your program. – More on functions later...
Example of a Nested Loop

• You want to collect the total grades of 100 students in a class

• Each student has multiple scores
  – Example: multiple homeworks, multiple quizzes, etc...

• You go through each student – one at a time – and get their scores
  – You calculate a sub-total grade for each student

• Then after collecting every student score, you calculate a grand total grade of the whole class and a class average (grand total / no. of students)
int students(100);
double grade(0), subtotal(0), grand_total(0);
for (int count = 0; count < students; count++) {
    cout << "Starting with student number: " << count << endl;
    cout << "Enter grades. To move to the next student, enter a negative number.\n"
    cin >> grade;
    while (grade >= 0) {
        subtotal = subtotal + grade;
        cin >> grade;
    } // end while loop
    cout << "Total grade count for student " << count << "is " << subtotal << endl;
    grand_total = grand_total + subtotal;
    subtotal = 0;
} // end for loop

cout << "Average grades for all students= " << grand_total / students << endl;
YOUR TO-DOs

- Finish reading up to (& including) Chapter 3
- Finish Lab1 by TOMORROW AT NOON (Fri, 10/6)

- HW2 is now ready
- Visit Prof’s and TAs’ office hours if you need help!

- Eat all your vegetables
</LECTURE>