Riddles

- What gets wetter and wetter the more it dries?
- You throw away the outside and cook the inside. Then you eat the outside and throw away the inside. What did you eat?
- What can you catch but not throw?
- What goes around the world but stays in a corner?
- Give me food, and I will live; give me water, and I will die. What am I?
- What have I got in my pocket?

Introduction to Programming Languages c++ Algorithms

Luke Sathrum – CSCI 20

Today's Class

- Machine, Assembly and High-Level Languages
- Advantages of High-Level Languages
- History of C / C++
- Relationship between C and C++
- Algorithm Definition
- Writing an Algorithm
- Pseudocode Definition
- Translating Pseudocode to Code

Introduction to Programming Languages

Programming Languages

- Computers are great tools
- Very limited on what they understand
  - Have a finite set of instructions that they understand
- These instructions are executed in binary
  - EX: `10110000 01100001`
- Executed by the Central Processing Unit (CPU)
- Different families of CPUs have different instructions that they understand

Programming Languages - Machine

- It is very hard to write a program in binary
  - Called **Machine Language**
  - This is how all the first programs were written
- **Source Code** is the collection of computer instructions
- Each program had to be written for a particular CPU family
  - Running on 2 different types of CPUs required you to rewrite your code for each one

Programming Languages - Assembly

- **Assembly Language** was invented to address the complexity of machine language
- Each instruction (1's and 0's) is identified by a short name
  - Has a 1-1 relationship to Machine Language
- This makes it much easier to write
- A computer cannot understand assembly directly
  - Needs an **assembler** to translate it to Machine Language
- Assembly programs written for one CPU family will not run on another

Programming Languages – Machine and Assembly

Machine Language	Assembly Language
`10110000 01100001`	`mov al, 061h`

- Here we are telling the CPU to put the hexadecimal value `61` into the `al` storage location

Programming Language – High-Level

- We call machine and assembly **Low-Level Languages**
  - They work with the nuts and bolts of the computer… binary
- High-Level Languages were created to address concerns
- These languages must be translated into a form the CPU can understand
- One line of a High-Level Language can be many lines of machine/assembly

Programming Languages – High-Level

- Examples of programming languages
  - C
  - C++
  - Pascal
  - Ada
  - Java
  - JavaScript
  - Python
  - Perl

Language Review

- Computers only understand binary
- Low-Level Languages
  - Machine
  - Assembly
- High-Level Languages
  - Multitude of ones out there
- Difference
  - 1-1 relationship to CPU instructions for Low-Level
  - 1-many relationship to CPU instructions for High-Level

High-Level Languages

Translating High-Level Languages

- Your CPU cannot natively read High-Level Languages
  - You must first translate it into Machine Language
- There are two ways of doing this
  - **Compiling**
  - **Interpreting**
- Both do the same thing, making your code readable by the computer
- Initially code was less efficient than Low-Level Languages
  - Now they are at least as good, if not better

Compiling

- Use a **compiler** to make your code readable by a CPU
- Reads code
- Produces stand-alone executable
- Once you have the executable the compiler is no longer needed

Interpreting

- Use an **interpreter** to read and execute your code all at once
- Much easier to write/create as they can be written in a High-Level Language
- Usually slower as they have to compile the code every time the program is run
- You always need the interpreter to run the program

Compiling vs. Interpreting

- Any language can be compiled or interpreted
- Usually the following are compiled
  - C, C++, Pascal
- Usually the following are interpreted
  - Perl, Javascript,
  - Called **scripting** languages
- Some languages use a mix of the two
  - Compiled to bytecode which is then interpreted
  - Java, Python

Why High-Level Language?

- Easier to write
- Requires less instructions to perform the same task

Machine	Assembly	C++

0004 8B45F8
0007 01C0
0009 83C005
000c 8945FC
000f B8000000
	`movl -8(%rbp), %eax`
	`addl %eax, %eax`
	`addl $5, %eax`
	`movl %eax, -4(%rbp)`
	`movl $0, %eax`
		`a = b * 2 + 5;`

- No need to keep track of registers
  - Where data is stored in the CPU
- Portable to different CPUs

High-Level Language Review

- High-Level Languages must be translated
- Can be either compiled or interpreted (or a combination)
- High-Level Languages have many advantages over Low-Level Languages

Introduction to C++

History of C

- The C Language was developed in 1972
- Developed primarily as a systems programming language
  - To write Operating Systems
- Goals
  - Easy to compile
  - Efficient access to memory
  - Efficient code

History of C

- The UNIX operating system was rewritten in C in 1973
  - This led to C becoming more popular
  - Showed C code was very portable
- Became standardized in 1989
  - Allowed for easier compilation of C code

History of C++

- An extension of the C programming language
- Originally called “C with Classes”
- Developed in 1979 by Bjarne Stroustrup
- Adds new features to the C language
  - Namely Object-Oriented methods
- Mostly a superset of C
  - Most C programs can run in C++
  - Not all C++ programs can run in C

C / C++ Review

- C was created in 1972
- Popular due to integration with UNIX
- C++ is a superset of C

Sample Code

- We'll create code to output "Hello World" to the screen

Algorithms and Pseudocode

Algorithms

- What is an **Algorithm**?
  - A set of instructions for solving a problem
- Everyday Examples?
  - Tie your shoe
  - Shower
  - Get Gas
  - Brush your Teeth
- In life there are different ways to solve the same problem

Algorithms and Programming

- Benefits
  - Know what you're doing before you do it
  - Save time programming
  - Get the correct answer
- Always a good idea to write your algorithm before you write your code
  - **Think** first
  - What is the program supposed to do?
  - What input do I need from the user?
  - What output does my program produce?

Creating an Algorithm

- Let’s solve the simple problem of summing (adding) two numbers we get from a user
- We will need to write it step-by-step
- We can’t assume we have anything yet
- Need to start at the beginning

Algorithm: Adding 2 Numbers Together

1. Get number 1 from user and store it
2. Get number 2 from user and store it
3. Add number 1 and number 2 and store the result
4. Output the result to the user

Pseudocode

- These steps we have written are the basis of **pseudocode**
- Allows us to describe how an algorithm (or our entire program) will work
- Allows us to design at a higher level than our code
- Allows us to use the pseudocode to easily write our code
- The pseudocode will become the basis for our **comments**
  - Comments are a way for us to describe our code to ourselves and other users

Summary

- Algorithms are just a set of instructions
- We write these instructions step-by-step
- Pseudocode is how we represent these algorithms in English
- Pseudocode becomes the basis for our comments

Review

- Two types of programming languages
  - The difference?
- Two ways of translating a high-level language
  - The difference
- Why high-level language?
- What is an algorithm?
- What is pseudocode?