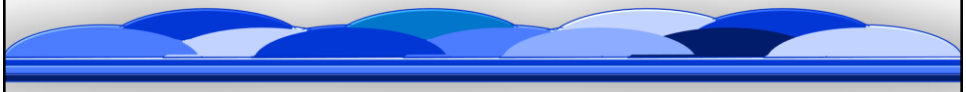
Chapter 2 – CS1A Review – P1



Programming Basics

Announcement

LAB 1

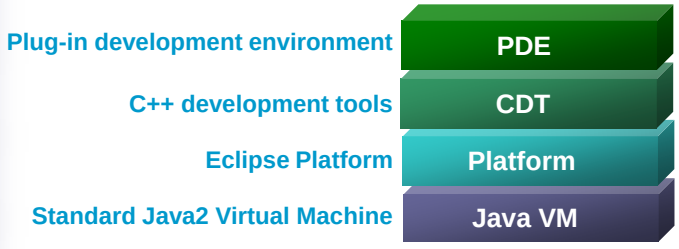
- Go through the eclipse tutorial → note the special instructions on the lab
- → Must have output

What is Eclipse?

Integrated Development Environment (IDE)

- A Universal Platform for Development Tools

- Open, extensible architecture based on plug-ins
- Open-source
  - see the Eclipse Project at Eclipse.org
- Multi-platform, multi-language, multi-vendor
- Endorsed by major tool vendors
- Reduced Complexity through customizable perspectives and views
- Support for popular features through open standards



Note:
It is required that you use Eclipse in this course.

Origins of C++

- Ken Thompson
  - Developed Unix
  - Used Assembly and "B" to program Unix
  - B → Derived from BCPL
- Denis Ritchie
  - 1970s, Developed C to program Unix
  - C was derived from B
- C was not as strictly typed as other languages
  - Allowed it to be more flexible
  - Easier to read and write than Assembly
  - But… more error prone → lacked automated checks

Origins of C++ (2)

- Bjarne Stroustrup
  - 1980s, Bell Labs developed C++
  - Based on C
    - If you can understand C you can understand C++
      - **Vise versa is not necessarily true**
    - C++ allows for object oriented programming (OOP)
      - **More modern style of programming**
    - Also, has stronger type checking and standards
      - **Easier to code**
      - **Easier to reuse**
      - **Easier to modify**
      - **Easier to debug**

Definitions

- Computer Programming
  - The process of implementing algorithms using a language the computer can understand
- Computer Program
  - An implementation of an algorithm
  - A step by step set of instructions performed by the computer
- High-level languages
  - "English" or "people friendly" languages
- Compiler
  - Interprets High level languages into machine language
  - c++ is a compiled language

What is a Programming Language

- Programming Language Consists of
- A set of special words, symbols and rules used to construct a program
  - Syntax – rules that dictate how valid instructions are written.
  - Semantics – rules that dictate the meaning attached to the instructions

Matters of Style

- Look over the lecture notes posted
  - For more info you can check out chapter 1
- Why have style?
  - Readability
  - Reusability
  - Modifiability
  - Easier to debug!

Note:
I won't be covering this in class → you need read the notes posted on the website and read the chapter for flowchart guidelines. CREATE A TEMPLATE
Let me know if you have any questions.
You will be expected to adhere to these guidelines as well as the flowchart guidelines

Storing data in Memory

- Once you give a place in memory a specific name you can refer to that location by using that name (the identifier)
  - →symbolic referencing

Identifiers

- a descriptive name that maps to a location in the computers memory
  - Identifiers should have meaningful names
- We have identifiers so we can
  - Retrieve data
  - Reuse data
  - Modify data

2 types of Identifiers

- Variables - contains data values that may change during program execution
  - The amount of memory to be reserved is determined at compile-time
  - Value is determined at runtime
- Named Constants - contains data values that can't be changed during program execution
  - The amount of memory to be reserved is determined at compile-time
  - Value is determined at compile-time
  - The value must be declared

Declaring Identifiers

- The compiler needs to know two things
  1 - The type of data that needs to be stores
    - How the contents of memory need to be viewed
    - C++ is strongly typed
      - **You have to be specific about what you are storing**

  2 - How much memory is needed to store your data

- In C++ the data type provides both pieces of information

Syntax	Description	Size	Data Values	Examples
int	integer	4 bytes	+ or - integers (whole #s), -2,147,483,648 to 2,147,483,647	3, 4, 235, 12152, -23,
char	character	1 byte	Characters enclosed in single quotes	'a','z','d', 'f'
float	floating point number	4 bytes	Pos. or neg. decimal numbers - including fractional part (up to 7 digits)	32.2,-23.32, 0.0, 123.2332
short	short integer	2 bytes	Pos. or neg. integers (whole numbers), -32,768 to +32767	Same as int
long	long integer	4 bytes (sometimes 8)	Same as int	Same as int,
double	double precision float	8 bytes	Floats up to 15 digits	Similar to float - larger #s
bool	Boolean	1 byte	One of 2 values: True or false	true, false

Using unsigned before an integer
- stores only positive values (including 0)
- doubles the value of positive integers you can store
- eg. unsigned short can store 0 - 65535

Identifiers Naming Rules

- Required rules →
  - Identifiers can only have
    - Letters
    - Numbers
    - Underscore (_)
  - must begin with a letter
  - Can't have spaces
  - Can't have special characters
- Remember they are case-sensitive

Identifiers – Naming Conventions

Stylistic rules → what we care about

ALWAYS Use meaningful names

Variables

- For 1 word → Keep it lowercase
- For 2 or more words
  - Begin with a lower case letter and only the first letter of each successive word will be capitalized

Constants

- All words in caps
- Use underscore to separate words
- eg TAX_RATE, PROGRAMMER

Can't use KEYWORDS

KEYWORD - a word that has some sort of predefined meaning in the context of a programming language

Declaring an Identifier

Variables

Syntax

type variableName;

Examples:

```
int sum;
float average;
char response;
```

The semi-colon tells the compiler that you are finished with the statement

Constants

Syntax

const type CONSTANT_NAME = value;

You must provide the value

Examples:

```
const int DAYS_IN_WEEK = 7;
const float SALES_TAX_RATE = 0.075;
```


Strings are special

Syntax

char *variableName*;

A simple character definition contains one single character value

- char singleChar;
- Can contain values such as

  'a', '!', 'C', '$', 'x', 'X'

Note the Single Quotes

If you have more than 1 character we need to store we use c-strings

Strings are special

C-Strings

- An array of characters
- The last character is called a null terminator (\0)
  - Tells the compiler when the string is ended
- We need to specify how many characters we want

Syntax

char *variableName*[*size*];

For Example

char lastName[15] → allocates space for 15 chars (14 + the null terminator)

char lastName[3] → allocates space for 3 chars (2 + the null terminator)

Data Table

- Describes what 2 things with comments?
  - What the variable represents
  - How the value is obtained

```
int ageOne;          // INPUT - first age from user
int ageTwo;          // INPUT - second age from user
char answer;         // INPUT - holds 'Y' or 'N' response
                     //         from user
char userName[20];   // INPUT - name of program user
float averageAge;    // CALC & OUT - avg. of two input
                     //         ages
```

Note:
This is the only time you'll have comments in line with the code

Assignment Statements

Syntax

variableName = *expression*;

Assigns the value of the expression (on the right) to the variable (on the left).

Example:

```
ageOne      =15;
ageTwo      = 23;
averageAge = (ageOne + ageTwo) / 2.0;
answer      = 'y';
```