Data Structures and Algorithms in C++

Class Meeting

Robot and Smart Systems
Manara University

Fall 2022

Instructor: Iyad Hatem

Course Objectives

- The primary goal of this class is to learn a useful subset of C++ programming language and fundamental data structures and algorithms expressed in C++.
- You will learn best practices for developing reliable software.
- You will acquire software development skills that are valued by employers.

Not Course Objectives

- Complete knowledge of C++
  - C++ is a huge, complex language!
  - The class will hit the important features.
  - You can learn the rest by yourself from online tutorials or the textbooks.
  - We will briefly touch the new features of C++ 11 and 14.
- Advanced data structures and algorithms
- Advanced algorithm analysis

Required Textbooks

- **Problem Solving with C++**, 10th edition
  - Author: Walter Savitch
  - Publisher: Pearson, 2017
  - ISBN: 978-0134448282

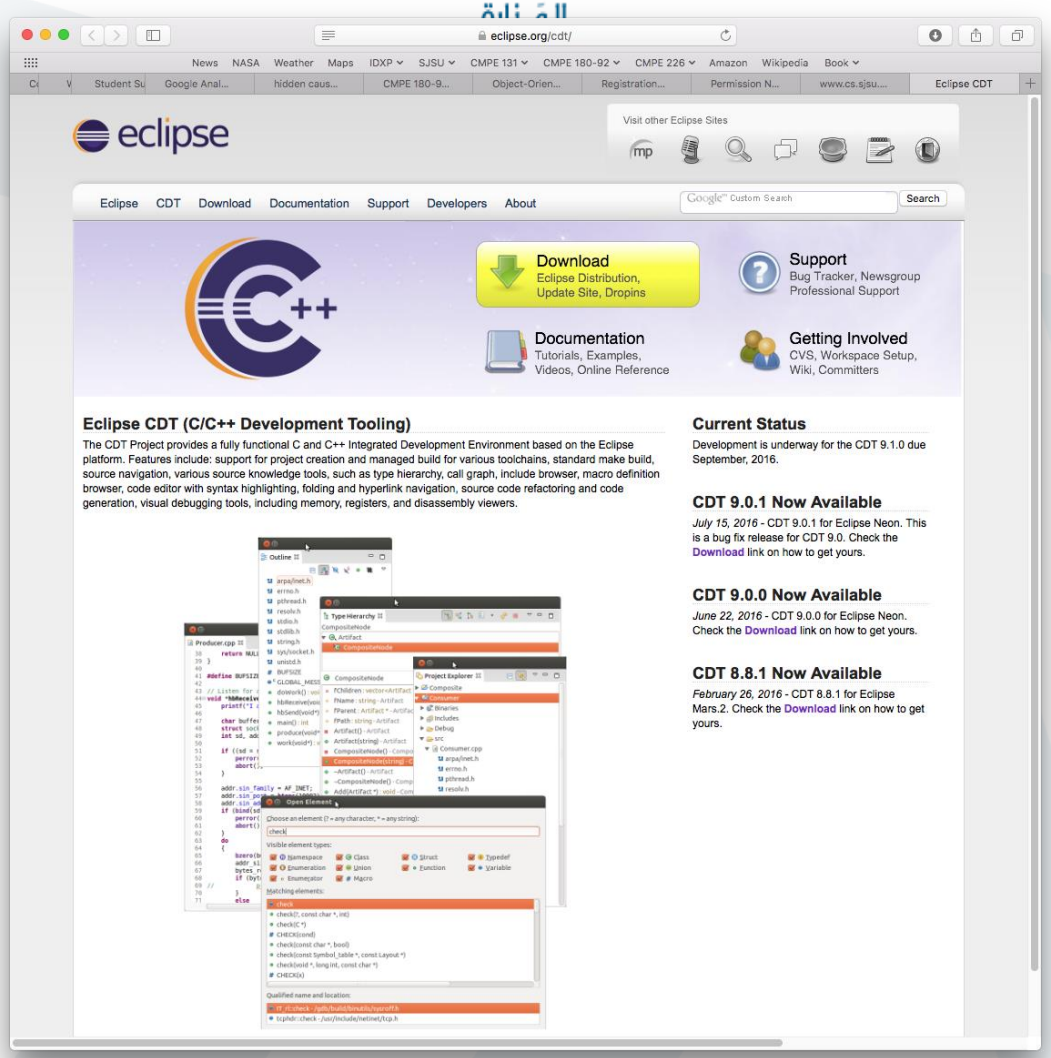
- **Data Structures Using C++**, 2nd edition
  - Author: D.S. Malik
  - Publisher: Cengage Learning, 2010
  - ISBN: 978-0324782011

You are responsible for doing the chapter readings before each class, as indicated in the class schedule.

Software to Install

- Install an integrated development environment (IDE) for C++ development on the Mac or Linux platform, such as:
  - Eclipse CDT (C/C++ Development Tooling): https://eclipse.org/cdt/
- You can choose your favorite IDE.

Software to Install, *cont'd*



C++ on the Mac and Linux Platforms

- GNU C++ is usually pre-installed on the Mac and Linux platforms.
- No further action required!
-

C++ on Windows 10

- The Windows platform has proven to be problematic for this class.
  - Difficult to install the Cygwin environment correctly.
  - Difficult to install C++ libraries successfully.
  - Serious compatibility challenges.

- Avoid using Microsoft's Visual C++ on Windows for this class.
  - You run the risk of writing programs that will not port to other platforms.

C++ on Windows 10, *cont'd*

- Install the Windows Subsystem for Linux (WSL).
  - See https://docs.microsoft.com/en-us/windows/wsl/install-win10
- Recommended: Install the Ubuntu distribution.
  - See https://www.microsoft.com/en-us/p/ubuntu/9nblggh4msv6?activetab=pivot:overviewtab

We will not provide support for Windows.

If you insist on running Windows,
you are on your own!

Useful Tutorials

- “Install Ubuntu on Windows 10 and on VirtualBox”
  - http://www.cs.sjsu.edu/~mak/tutorials/InstallUbuntu.pdf
- “Configure Ubuntu for Software Development”
  - http://www.cs.sjsu.edu/~mak/tutorials/ConfigureUbuntu.pdf
- “Install Eclipse for Java and C++ Development”
  - http://www.cs.sjsu.edu/~mak/tutorials/InstallEclipse.pdf

C++ 2011 Standard

- We will use the 2011 standard version of C++.
- You must set this standard explicitly for your project in Eclipse or your chosen IDE.
- On the command line:

```
g++ foo.cpp --std=c++11 -o foo
```

Two hyphens!

Set the C++ 2011 Standard in Eclipse

- Right-click on your project in the project list at the left side of the window.
- Select “Properties” from the drop-down context menu.
- In the left side of the properties window, select “C/C++ Build” ➔ “Settings”.
- In the Settings dialog, select “GCC C++ Compiler” ➔ “Dialect”.
- For “Language standard” select “ISO C++ 11”.
- Click the “Apply” button, answer “Yes”, and then click the “OK” button.

Remember to do all these steps for every C++ project in Eclipse.

Assignments

- You will get lots of programming practice!
  - A main programming assignment each week.
  - Several small practice programs that emphasize specific skill needed to solve the main assignment.
- We will use the online CodeCheck system which will automatically check your output against a master.
  - You will be provided the URL for each assignment.
  - You can submit as many times as necessary to get satisfactory output.

Assignments, *cont'd*

- Assignments will be due the following week, before the next lecture.
- Solutions will be discussed at the next lecture.
- Assignments <u>will not be accepted</u> after solutions have been discussed in class.
  - Late assignments will receive a 0 score.

Individual Work

- You may study together.
- You may discuss the assignments together.

- But whatever you turn in must be your individual work.

Academic Integrity

- Copying another student's work or sharing your work is a violation of academic integrity.
- Violations will result in harsh penalties by the university.
  - Academic probation.
  - Disqualified for TA positions in the university.
  - Lose internship and OPT sponsorship at local companies.
- Instructors are obligated to report violations.

Moss

- Department policy is for programming assignments to be run through Stanford University's Moss application.
  - Measure of software similarity
  - Detects programming plagiarism
  - http://theory.stanford.edu/~aiken/moss/
- Moss is not fooled by
  - Renaming variables and functions
  - Reformatting code
  - Re-ordering functions

Example Moss output:
http://www.cs.sjsu.edu/~mak/Moss/

Exams

- The midterm and final examinations will be open book and conducted online.
- Instant messaging, e-mails, texting, tweeting, file sharing, or any other forms of communication with anyone else during the exams violates academic integrity.

Exams, *cont'd*

- There can be no make-up midterm examination unless there is a documented medical emergency.
- Make-up final examinations are available only under conditions dictated by University regulations.

Final Class Grade

- 65% assignments
- 15% midterm
- 20% final exam

- The class is graded CR/NC.
  - Students who have a weighted score above the passing threshold at the end of the semester will receive the CR grade.

- We expect least 80% of students will pass.
  - In some past semesters when I've taught this class, the pass rate has been higher than 95% in the past.

Take Roll

Fast Pace!

- This class will move forward at a fast pace.
- Lectures will consist of:
  - New PowerPoint slides by the instructor
  - PowerPoint slides from the textbook publishers
  - Program examples and live demos
  - Questions, answers, and discussion
- Lecture materials will be posted to the class webpage: http://www.cs.sjsu.edu/~mak/CMPE180A/index.html

Discussion Forum

- Please use the Discussions feature of Canvas.
  - Ask questions
  - Answer questions
  - Chat

- If you have a question, please ask it in the Discussions feature .
  - Others may have the same question.
  - I'll only have to answer the question once.
  - Other students can provide answers before I do.

What is C++

- An object-oriented programming (OOP) language.
  - Supports encapsulation, inheritance, polymorphism.
  - Based on the C language with added OOP features.
- A powerful but complex language!
  - Lots of features.
  - Somewhat arcane syntax.
  - Easy to make programming errors.
  - Things happen automatically at run time that you may not expect.

A Useful Subset of C++

- We will only learn a useful subset of C++.
  - Very few people (not including your instructor) know the entire language.
- Among professional C++ programmers, everybody knows a different subset, depending on experience, training, and application domains.

What Happened?

- We may have to figure out together what happened when …
  - You've accidentally stumbled onto an obscure language feature.
  - Your program runs slower than expected.
  - Your program mysteriously crashes.
- Your program may appear to run fine on your machine but then crash in CodeCheck.
  - It's usually because your program attempted to access protected memory via a bad pointer.

Our First C++ Program

- The infamous “Hello, world!” program.

helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, world!" << endl;
    return 0;
}
```

- Compiled and run on the command line:

```
~/programs/HelloWorld: g++ helloworld.cpp --std=c++11 -o helloworld
~/programs/HelloWorld: ./helloworld
Hello, world!
```

Algorithms and Program Design

Savitch_ch_01.ppt: slides 57– 60

- Display 1.4
  - Compiling and Running a C++ Program
- Display 1.5
  - Preparing a C++ Program for Running
- Display 1.7
  - Program Design Process

Sample C++ Program: Pods and Peas

Savitch_ch_01.ppt: slides 34 – 44

- “A Sample C++ Program”

Savitch_ch_01.ppt: slide 61

- Display 1.8
  - Pods and peas program

Identifiers and Variables

- Identifiers are names.
- Variables represent values that can change.
  - Variables have names (variable identifiers).
- Declare variables before you use them.
  - A declaration tells what is the variable's datatype (integer, float, double, character, boolean, etc.).
  - A declaration can also give an initial value to the variable.

```
int n;
double ratio;
bool is_prime;
char ch;
string name;
```

```
int length = 0;
double temp = 98.6;
string name = "Frank";
```

Keywords

- Keywords are reserved by C++ and you cannot use them as identifiers.
  - Examples: `if for while`

Assignment Statements

- At run time, be sure to initialize a variable (give it a value) before you use it.
  - Either initialize the variable when you declare it.
    - Example: `int i = 5;`
  - Or execute an assignment statement.
    - Example: `i = 10;`
- Do not confuse = (assignment) with == (equality comparison).

```
i = 10;          // assign the value of 10 to variable i
if (i == 10)     // test whether or not i is equal to 10
```

Break

Output Stream

- Values written by the program at run time.
- Standard output stream: **`cout`**
  - Default: the display
- Example:

```
cout << "x equals " << x << endl;
```

insertion operator

  - Insert (write) the string "**`x equals`**" followed by the value of variable **`x`** followed by a carriage return (**`endl`**) to the display.

Formatting Real Numbers for Output

- Call methods of **`cout`** to format real numbers.
- **`cout.setf(ios::fixed);`**
  - Use fixed-point notation (not scientific).
- **`cout.precision(2);`**
  - How many places after the decimal point (e.g., 2).
  - You can also write:

```
#include <iostream>
#include <iomanip>
...
cout << fixed << setprecision(16);
```

Input Stream

- Data read by the program at run time.
- Standard input stream: **`cin`**
  - Default: the keyboard
- Example:

```
cin >> x >> y;
```

extraction operator

  - Extract (read) the next two values from the keyboard and assign the values to **`x`** and **`y`**, respectively.

Input From `cin`

- `cin >> v1 >> v2 >> v3;`
  - Read values into multiple variables.
  - The input values should be separated by one or more spaces.
- The values are not read until you press the return key.
  - Therefore, you can backspace and make corrections.

`#include` and `using namespace`

- `#include <iostream>`
  - Include the definitions of `cin` and `cout` in your program.
- `using namespace std;`
  - Make the standard namespace `std` available to the program.
  - The names `cin` and `cout` and other important names reside in the standard namespace.

Some Basic Data Types

- A datatype (also: data type) determines
  - what kind of data values
  - what operations are allowed
- Data type **`int`** for integer values without decimal points.
  - Examples: **`0 2 45 -64`**
- Data type **`short`** for small integer values.
- Data type **`long`** for very large integer values.

Some Basic Data Types, *cont'd*

- Data type **`double`** for real numbers.
  - Fixed-point notation: **`34.1 23.0034 -1.0 89.9`**
  - Scientific notation: **`3.67e17 5.89E-6 -7.23e+12`**
- Data type **`float`** for less precision and smaller magnitude.
- Data type **`char`** for individual characters.
  - Examples: **`'a'  'Z'`**
  - Use only single quotes for character constants in a program.

Some Basic Data Types, *cont'd*

- Data type **`bool`** for the Boolean values **`true`** and **`false`**.
- The Boolean value **`false`** is stored as the integer 0.
- The Boolean value **`true`** is stored as the integer 1.

`cin` Skips Input Blanks

- The statements

```
char ch1, ch2;
cin >> ch1 >> ch2;
```

when given the input

```
A  B
```

will set **`ch1`** to **`'A'`** and **`ch2`** to **`'B'`**.

`cin` uses blanks and line feeds to separate input data values, but otherwise it skips the blanks and line feeds.

String Type

- `#include <string>`
  - Required if your program uses strings.
- Enclose string values with double quotes in your program.
  - Example: `"Hello, world!"`
- To input a string from `cin` that includes spaces, all in one line:

```
string str;
getline(cin, str);
```

Type Compatibilities and Conversions

- `int pi = 3.14;`
  - `double` → `int` is invalid. You cannot set a `double` value into an `int` variable .

- Some valid conversions:
  - `int` → `double`
  - `char` → `int`
  - `int` → `char`
  - `bool` → `int`
  - `int` → `bool`

Any nonzero integer value is stored as true. Zero is stored as false.

Arithmetic

- Arithmetic operators: `+ - * / %`
- Integer `/` result if both operands are integer.
  - Quotient only.
  - Example: The value of `11/3` is 3.
- Use the modulo operator `%` to get a remainder.
  - Example: The value of `11%3` is 2.
- Double `/` result (includes fractional part) if either or both operands are double.

Operator Shorthand

- `n += 5` shorthand for `n = n + 5`
- `n -= 5` shorthand for `n = n - 5`
- `n *= 5` shorthand for `n = n*5`
- `n /= 5` shorthand for `n = n/5`
- `n %= 5` shorthand for `n = n%5`

The `if` Statement

- Example `if` statement:

```
if (n <= 0)
{
    cout << "Please enter a positive number." << endl;
}
```

- Example `if else` statement:

```
if (hours > 40)
{
    gross_pay = rate*40 + 1.5*rate*(hours - 40);
}
else
{
    gross_pay = rate*hours;
}
```

`while` Loops

- Example `while` loop:

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

- Example `do while` loop:

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

Named Constants

- It's good programming practice to give names to constants:

```
const double PI = 3.1415626;
```

- Easier for humans to read the program.
- Easier to modify the program.
- Convention: Use ALL_CAPS with underscores if necessary for the names of constants.

Boolean Operators

- Relational operators: `==` `!=` `<` `<=` `>` `>=`
- And: `&&`
- Or: `||`
- Not: `!`

- Short-circuit operation: `p && q`
  - `q` is not evaluated if `p` is false

- Short-circuit operation: `p || q`
  - `q` is not evaluated if `p` is true

Precedence Rules

Savitch_ch_03.ppt: slides 8-13

Precedence Rules

The unary operators +, –, ++, --, and !.
The binary arithmetic operations *, /, %
The binary arithmetic operations +, –
The Boolean operations <, >, <=, >=
The Boolean operations ==, !=
The Boolean operations &&
The Boolean operations ||

Highest precedence (done first)

Lowest precedence (done last)

Enumeration Types

- A data type with values defined by a list of constants of type **`int`**
  - Examples:

```
enum Direction {NORTH, SOUTH, EAST, WEST};

enum MonthLength{JAN_LENGTH = 31,
                 FEB_LENGTH = 28,
                 MAR_LENGTH = 31,
                    …
                 DEC_LENGTH = 31};
```

Nested `if` Statements

- Example:

```
if (net_income <= 15000)
{
    tax_bill = 0;
}
else if ((net_income > 15000) && (net_income <= 25000))
{
    tax_bill = (0.05*(net_income - 15000));
}
else // net_income > $25,000
{
    five_percent_tax = 0.05*10000;
    ten_percent_tax = 0.10*(net_income - 25000);
    tax_bill = (five_percent_tax + ten_percent_tax);
}
```

The `switch` Statement

- Use a **`switch`** statement instead of nested **`if`** statements to compare a single integral value for equality.
  - Note the need for the **`break`** statements.
  - Note the **`default`** case at the bottom.

```
int digit;
...
switch(digit)
{
   case 1: digit_name = "one";   break;
   case 2: digit_name = "two";   break;
   case 3: digit_name = "three"; break;
   case 4: digit_name = "four";  break;
   case 5: digit_name = "five";  break;
   case 6: digit_name = "six";   break;
   case 7: digit_name = "seven"; break;
   case 8: digit_name = "eight"; break;
   case 9: digit_name = "nine";  break;

   default: digit_name = ""; break;
}
```

The Increment and Decrement Operators

- `++n`
  - Increase the value of `n` by 1.
  - Use the increased value.
- `n++`
  - Increase the value of `n` by 1.
  - Use the value before the increase.

The Increment and Decrement Operators, *cont'd*

- `--n`
  - Decrease the value of `n` by 1.
  - Use the decreased value.
- `n--`
  - Decrease the value of `n` by 1.
  - Use the value before the decrease.

for Loops

- Example:

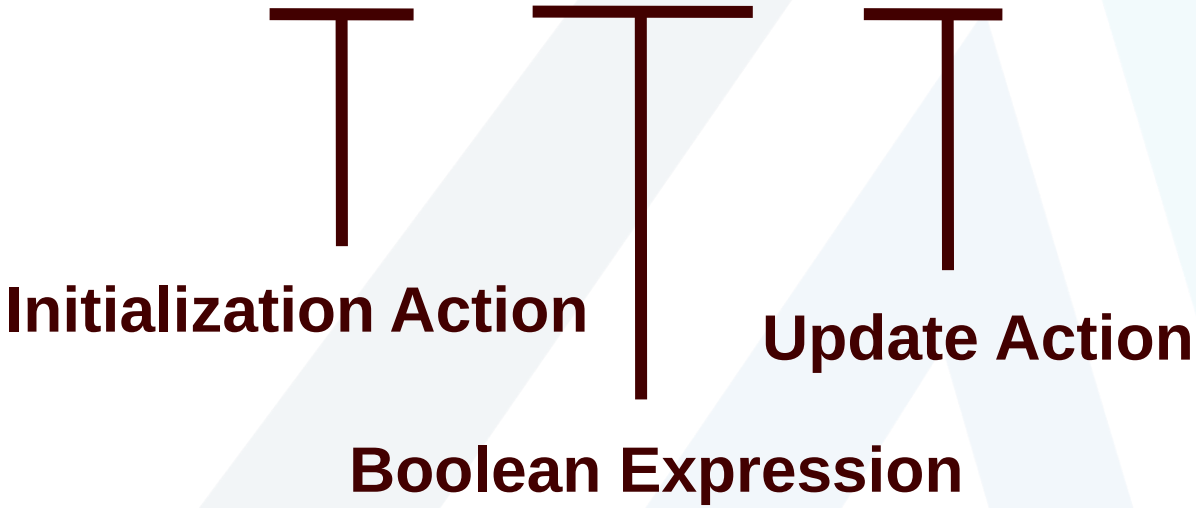
```
int sum = 0;

for (int n = 1; n <= 10; n++)
{
    sum = sum + n;
}

cout << "The sum of the numbers 1 to 10 is "
     << sum << endl;
```

Note that variable **n** is local to the loop body.

`for` Loops, *cont'd*

- The **`for`** loop uses the same components as the **`while`** loop, but in a more compact form.

```
for (n = 1; n <= 10; n++)
```



The `break` Statement

- Use the **`break`** statement to exit a loop before "normal" termination.
- Do not overuse!
  - Well-designed loops should end normally.
- This use of **`break`** in a **`for`** statement is different from the necessary use of **`break`** in a **`switch`** statement.

Nested Loops

- If you have an “outer loop” that contains an “inner loop”, then for each iteration (execution) of the outer loop, the inner loop goes through all of its iterations.
- This concept extends to more than just one loop inside another.
  - Loops can nest deeply, although usually there are no more than three loops.
- Nested loops are a very common in programs.

Nested Loops, *cont'd*

nestedloop.cpp

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    for (int i = 1; i <=2; i++)
    {
        for (int j = 9; j <= 12; j++)
        {
            cout << "i = " << i
                 << ", j = " << j << endl;
        }
    }

    return 0;
}
```

```
i = 1, j = 9
i = 1, j = 10
i = 1, j = 11
i = 1, j = 12
i = 2, j = 9
i = 2, j = 10
i = 2, j = 11
i = 2, j = 12
```

Loop Considerations

- Choosing the right kind of loop to use
- Designing loops
- How to control a loop
- How to exit from a loop
- Nested loops
- Debugging loops

Savitch: Chapter 3