General Information:

Instructor	: Omar Mansour, Ph.D.
Email	: omansour@nyu.edu
Office	: 370 Jay street, room 844
Office hours	: TBD, or by appointment.
Credit Hours	: 4
Class sessions :	Sec A: 12:30 PM - 1:50 PM, Prof. Mansour, Jacobs 473
	Sec D: 3:30 PM - 4:50 PM, Prof. Mansour, 2 MetroTech #907
	Sec B: 9:30 AM – 10:50 AM, Prof. Sterling
	Sec C: 11:00 AM – 12:20 PM, Prof. Sterling

Teaching Assistants:

Thaison Le tnl2012@nyu.edu
Anna Tanaka Viertler at5076@nyu.edu
Yirong Wang yw5490@nyu.edu
Alexander Wu azw7225@nyu.edu
Dorothy Zhang zz2953@nyu.edu
Yufei Zhen yz9579@nyu.edu

Required Text Book:

N/A, just the lecture notes

Course Description

This intermediate-level programming course teaches object-oriented programming in C++. Topics: Pointers, dynamic memory allocation and recursion. Classes and objects including constructors, destructors, methods (member functions) and data members. Access and the interface to relationships of classes including composition, association and inheritance. Polymorphism through function overloading operators. Inheritance and templates. Use of the standard template library containers and algorithms.

Pre-requisites:

CS-UY 1134 (C- or better)

Course Objectives:

- Design and implement classes in C++
- Inheritance
- Use of pointers, dynamic memory and copy control
- Operator overloading
- Use of the STL
- Exception handling and assertions

Grading:

Homework and Labs	: 25%
Mid-term exam	: 30%
Final exam	: 45%
Class participation	

Grade Thresholds:

Grade letter	Cutoff percentage
A	93
A-	90
B+	87
B	83
B-	80

Grade letter	Cutoff percentage
C+	77
C	73
C-	70
D+	65
D	60

The midterm is scheduled during the common exam period:

March 12, 2024: 12:30-2pm. Content covered **through lecture #12**

Topics

Basics of C++: static typing, conditions, loops, functions, structs and vectors.
Basic OOP: encapsulation and delegation, methods, data hiding, constructors, const methods, nested types.
Pointers: addresses, pointer types, pointers and const, dynamic memory
Copy control: destructor, copy constructor and assignment operator.
Implementing the vector type: dynamic arrays, pointer arithmetic, index operator, explicit constructor
Cyclic Association
Separate compilation
Operator overloading
Inheritance
Iterators
Implementing linked lists
Recursion
Generic Programming: class and function templates
STL: collections, containers, algorithms, functors, lambda expressions
Exceptions and assertions

Lec	Topics	Date	Recitations
1	Intro, administrivia, goals, C++: types, conditions, loops, vector, I/O	1/22	Rec 01 Basic C++ Tutorial, Vectors, strings and functions
2	More C++ Basics: Functions, ranged for ref / const, struct	1/24	
3	Structs and Vectors	1/29	Rec 02 Structs and Vectors
4	OOP: classes, data hiding, encapsulation, constructor, const methods	1/31	
5	OOP: op<<, delegation, nested types	2/05	Rec 03 Classes: Encapsulation and Data Hiding
6	Pointers! Addresses, pointer types, address-of, dereference, this, nullptr	2/07	
7	->, const with pointers, dynamic memory	2/12	Rec 04 Pointer tutorial
8	Copy control: deep vs. shallow, destructor, copy constructor	2/14	
9	Copy control: Assignment operator. Vector class design	2/21	Rec 05 Pointers and Association
10	Implementing the Vector class, dynamic arrays, pointer arithmetic, push_back	2/26	Rec 06 Copy Control
11	Finish vector class: op[], explicit, ranged for support	2/28	
12	Operator Overloading	3/04	Rec 07 Operator Overloading
13	Cyclic association	3/06	
14	Separate compilation	3/11	Rec 08 Cyclic association and Separate Compilation
15	Inheritance: substitutability, slicing, polymorphism	3/13	
16	Inheritance: override, constructors, importance of interface,	3/25	Rec 09 Inheritance Tutorial
17	Inheritance: abstract, protected, hiding	3/27	
18	Inheritance: poly / non-member, overloading vs. overriding,	4/01	Rec 10 More Inheritance
19	Inheritance: inheritance with copy control	4/03	
20	Inheritance: polymorphism inside constructors, multiple inheritance; Singly linked list basics	4/08	Rec 11 Linked List Basics
21	More singly linked list basics	4/10	
22	Finish Linked lists, introduce Vector Iterators	4/15	Rec 12 Implementing a Linked List Class
23	Constant Vector iterators, Templates, STL	4/17	
24	STL	4/22	Rec 13 STL
25	Recursion	4/24	
26	Recursion	4/29	Rec 14 Recursion / Exceptions
27	Exceptions and Assertions	5/01	
28	Special Topics	5/06	
Final Exam - TBD			

Attendance and participation policy:

Attendance in lecture is **strongly** encouraged but not required.

Attendance in lab **is required**.

- Be on time. **Lateness will be penalized.**
- If you do come in late, i.e. after attendance has been called, **check in with the TA promptly**.
- You may leave early **if you have been checked out by a TA.**
- **Leaving early without being checked out will result in a zero for the lab**.

Homework

- All work is **to be done by you**.
  - Be careful that you don't get "too much" help from others.
  - Plagiarism on a homework assignment will result in significant penalties.
  - If anything about an assignment is confusing, ask *us*!
- assignments *are* accepted late, but with a penalty.
  - first day: 5%
  - second day: total of 10%
  - third day: total of 20%
  - fourth day: total of 40%
  - fifth day: total of 80%
  - No credit will be given after that.

Attendance and participation policy – cont.:

Lab assignments (weekly)

all lab assignments must be **turned in**, if you want to receive any credit.

may be checked out by a TA during lab to receive full credit (but **you still have to turn them in**!).

If not checked out during lab, then they will be graded by the TAs. (There are a few labs, for example the last lab of the semester, that must be checked out in lab by a TA.)

expected to be completed during lab, but we allow you to turn them in during the weekend without penalty.

not accepted after the weekend. Do not wait till the last minute to turn them in.

Academic Honesty:

- Students at NYU are expected to be honest and forthright in their academic endeavors.
- Academic dishonesty includes **cheating**, **unapproved collaboration**, coercion, inventing false information or citations, **plagiarism**, tampering with computers, destroying other people’s coursework, lab or studio property, theft of course materials, or other academic misconduct. If you have questions regarding this policy, contact your professor *prior* to submitting the work for evaluation. See your academic catalogue for a full explanation.
- All students must adhere to the NYU Tandon school of engineering’s “Student Code of Conduct”, https://engineering.nyu.edu/campus-and-community/student-life/office-student-affairs/policies/student-code-conduct

Academic Honesty (cont.):

- All assignments, unless otherwise explicitly listed, are to be done independently. Unless explicit permission from the instructor is provided, no outside sources may be used. If you have any doubts of your sources or their applicability, please contact the instructor as soon as possible.
- Anyone caught cheating in this course will receive a “0” on the assignment/assessment and the professor additionally retains the option of significantly reducing the final grade. If a student is caught a second time in, he/she shall fail the course.

Agenda

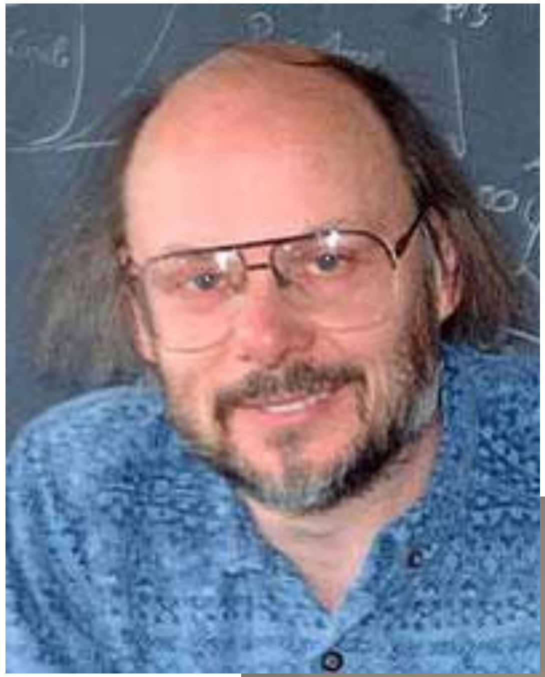
- Where to find more information
- Who created C++?
- Simple C++ program
- Hello world - writing to standard output
- Variables and types
- Getting data - reading standard input
- Conditions
- Logical operators
- Loops
- Collections: vectors and strings
- File I/O

Where to find more...

- http://cis.poly.edu/jsterling/cs2124/LectureNotes/01.Intro.html
- http://www.cplusplus.com
- http://cppreference.com

Who created C++?

- Bjarne Stroustrup
- AT&T Bell Labs
- Later:
  - University of Texas A&M
  - Morgan Stanley
  - Columbia University (visiting)
- Inspired by C and Simula
- http://www.stroustrup.com/



Simplest Program

```
int main() {
    return 0;
}
```

- Every program *must* have a function called **main**
- Blocks of code are surrounded by *braces*: **{}**
- Statements end with a semi-colon: **;**

- Every function *must* state what type it returns.
- main() must return an integer type, **int.**
  - The value zero means successful completion
- If a function returns anything, then it *must* have a return statement.

Simpler-est Program

```
int main() {
    //return 0;
}
```

- Actually, every function *other than* main that returns something must have a return statement,
- If main does not have a return statement then it will return 0. Stroustrup was being nice to us.
- Note that a comment begins with ***//***

Hello CS2124!

```
#include <iostream>

int main() {
    std::cout << "Hello CS2124!\n";
}
```

- **<<** is the output operator.
  - The target stream appears on the left, and what is printed is on the right
- **std::cout** represents standard output, i.e. by default, the screen
- String *literals* are always surrounded by *double* quotes: **" "**
- **#include** specifies a header file for compiler.
- iostream provide the definition for std::cout, among other things.

Hello CS2124! (take 2)

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

int main() {
   // std::cout << "Hello CS2124!\n";
   cout << "Hello CS2124!\n";
}
```

- using namespace std;
  - Allows us to refer to cout and many other symbols without having to type std::

Hello CS2124! (take 3)

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

int main() {
    // cout << "Hello CS2124!\n";
    cout << "Hello CS2124!" << endl;
}
```

- output can be *chained*
- endl: an end-of-line character.
  - Perhaps easier to type than: "\n"
  - Also “flushes” the output stream. That’s not crucial for us right now.

One key difference between C++ and Python?

- This is just a start, but…
- Every variable is declared to have a **type** and can only hold things of that **type**
  - char c;
  - int n;
  - unsigned u;
  - double d;
  - string s;
- This is also true of
  - function parameters
  - And function return types
- *Many* other languages are like this, such as C and Java and C#.
  - They are **statically-typed** languages

Variables

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

int main() {
    int x = 42;
    double d = 3.14159;
    string s = "the cat in the hat";
    cout << "x: " << x <<  ", d: " << d << ", s: " << s << endl;
}
```

- Every variable is declared to *have a type* and can *only* hold things of that type
- We needed the include for string because **strings are not primitive / built-in**

Uninitialized?

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int x;
    cout << "x: " << x << endl;
}
```

- What would the output be if we *don't* initialize an integer?
- **No guarantees**!!! This behavior is **undefined**.
- Your compiler *might* be nice and warn you;
- When run on Mac, x was zero.
- When run on Linux, not so lucky!

Uninitialized (2)

```
void foo() {
    int a = 17;
    cout << "a: " << a << endl;
}

void bar() {
    int b;  // Not initialized!!!
    cout << "b: " << b << endl;
}

int main() {
    foo();
    bar();
}
```

```
a: 17
b: 17
```

- b is using the same memory location as a
- **Your mileage may vary**!! (undefined behavior)

Getting input

```
int main() {
    int x = -1;
    cout << "x: " << x << endl;
    cout << "input an integer value: ";
    cin >> x;
    cout << "x: " << x << endl;
}
```

- **cin** is standard input, i.e. by default the keyboard
- Notice that the angle brackets for input “point” from the stream to the variable.

Conditions

```
int main() {
    int x;
    cout << "x? ";
    cin >> x;
    if (x == 6) {
        cout << "x is a small perfect number\n";
    }
    else if (x == 42) {
        cout << "x is the answer\n";
    }
    else {
        cout << "x is something else\n";
    }
}
```

- **if**, **else if**, and **else**
- The condition goes inside parentheses
- The code to handle the condition may go in a block (**a code block**), enclosed in curly braces.
    - Else, only the single proceeding line is part of the condition!

Logical Operations

```
int main() {
    int x;
    cout << "x? ";
    cin >> x;
    if (x == 6 || x == 28) {
        cout << "x is a small perfect number\n";
    }
    else if (x >= 0 && x <= 9) { // Note, no: 0 <= x < 10
        cout << "x is an imperfect single digit number\n";
    }
    else if (!(x < 10 || x > 99)) {
        cout << "x is a two digit number\n";
    }
    else {
        cout << "x is something else\n";
    }
}
```

- and: **&&**
- or: **||**
- not: **!**
- Do not use the **words:** and, or, not.

Loops: while

```
int main() {
    int x = 10;
    while (x >= 0) {
        cout << x << ' ';
        --x;
    }
    cout << endl;
}
```

- Condition must be in parentheses
- Code for the loop goes in a block
- Here we are using the pre-decrement operator.
  In Python we would write: x -= 1

Loops: for

```
int main() {
    for (int x = 10; x >= 0; --x) {
        if (x == 5) continue;  // yes, C++ has continue and break
        cout << x << ' ';
    }
    cout << endl;
}
```

- for loop has three parts within the parentheses, separated by semicolons
  - Initialization. Here x is being *initialized to* 10
  - Test for whether to enter the body of the loop. Here, only if x is greater than 0
  - Update. Here, we just need to decrement x
- Often more concise than a while loop.
- Also, note that the ***scope*** of x is limited to the for loop

Loops: do-while

```
int main() {
    int x = 10;
    do {
        cout << x << ' ';
        --x;
    } while (x > 0); // Remember the semicolon
    cout << endl;
}
```

- Always executes the body of the loop **at least once**.
- Consider **what would happen if x started out as -1**
- When you want this, it is very nice to have,
  but you will more often use the while and for loops.

Collections: vector

```
int main() {
    vector<int> v; // v can only hold integers
    cout << "v.size(): " << v.size() << endl;

    v.push_back(17);
    v.push_back(42);
    cout << "v.size(): " << v.size() << endl;

    for (size_t i = 0; i < v.size(); ++i) {
        cout << v[i] << ' ';
    }
    cout << endl;

    v.clear();
    cout << "v.size(): " << v.size() << endl;
}
```

- **#include <vector>**
- *Similar to* Python's list and Java's ArrayList
- *"Generic"* type
- Other handy methods: back(), pop_back(), capacity()

How can we print in reverse?