CS32 Discussion
2019 Summer - Week 1
Qi Zhao
zhaoq0563@ucla.edu
Jun-28-2019
Your TA

• Qi Zhao
• Email: zhaoq0563@ucla.edu
• Discussion session: 12pm – 2pm
• Office hours:
  • Monday: 4pm - 6pm
  • Location: Eng VI, Common study area outside Lab 397

• Practice is always important!
What is covered in a Discussion Session

• Review and practice everything that was covered during the lectures in the previous week

• Answer any questions that you could not ask during the lecture

• A high level review of the topics of the week with more focus on coding examples that illustrate those topics

• Practice by hand – You writing some code
Outline

- OOP revisit
- Debugging
- Dynamic memory allocation
Outline

• OOP revisit
• Debugging
• Dynamic memory allocation
Object-Oriented programming in C++

- Classes as units of encapsulation
- Information Hiding
- Inheritance
- Polymorphism and dynamic dispatching
- Storage management
- Multiple inheritance
Classes

• Encapsulation of type and related operations

```cpp
class point {
    double x, y; // private data members

public:
    point (int x0, int y0); // public methods
    point () { x = 0; y = 0; }; // a constructor
    void move (int dx, int dy);
    void rotate (double alpha);
    int distance (point p);
};
```
A class is a type: objects are instances

```plaintext
point p1 (10, 20);  // call constructor with given arguments
point p2;           // call default constructor
point p3();         // ?
```

Methods are functions with an implicit argument

```plaintext
p1.move (1, -1);    // special syntax to indicate object
```
Implementing methods

No equivalent of a body: each method can be defined separately

```cpp
void point::rotate (double alpha) {
    x = x * cos (alpha) - y * sin (alpha);
    y = y * cos (alpha) + x * cos (alpha);
}

// x and y are the data members of the object on which the
// method is being called.
// if method is defined in class declaration, it is inlined.
```
Constructors

• One of the best innovations of C++

• Special method(s) invoked automatically when an object of the class is declared
  
  point (int x1, int x2);
  point ();
  point (double alpha; double r);
  point p1 (10,10), p2; p3 (pi / 4, 2.5);

• Name of method is name of class

• Declaration has no return type.
The target of an operation

• The implicit parameter in a method call can be retrieved through this:

```cpp
class Collection {
    Collection& insert (thing x) {
        // return reference
        ... modify data structure
        return *this;  // to modified object
    }
};

my_collection.insert (x1).insert (x2);
```
Classes and private types

• If all data members are private, class is identical to a private type: visible methods, including assignment.
• A struct is a class with all public members
• How much to reveal is up to programmer
• Define functions to retrieve (not modify) private data

```cpp
int xcoord () { return x;};
int ycoord () { return y;};
p2.x = 15; // error, data member x is private
```
Destructors

- If constructor allocates dynamic storage, need to **reclaim** it

```cpp
class stack {
    int* contents; int sz;

public:
    stack (int size) { contents = new int [ sz = size];};
    void push ();
    int pop ();
    int size () { return sz;}; }

stack my_stack (100); // allocate storage dynamically
// When is my_stack.contents released?
```
If constructor uses resources, class needs a destructor

• User cannot deallocate data because data member is private: system must do it
  ~stack ( ) {delete[ ] contents;};

• Inventive syntax: negation of constructor

• Called automatically when object goes out of scope

• Almost never called explicitly
Initialization List

- Initializer List is used to initialize data members of a class
  - For primitive data members it is optional
  - For member objects which do not have default constructor, it is necessary

```cpp
class Weapon {
private:
    string weapon_name;
public:
    Weapon(string name) {
        weapon_name = name;
    }
};

class Soldier {
private:
    string m_name;
    int m_health;
    Weapon m_weapon;
public:
    Soldier();
};

Soldier::Soldier() {
    m_name = "Joe";
    m_health = 100;
    m_weapon = Weapon("gun");
}
```
Copy and assignment

point p3 (10,20);
point p5 = p3; // component-wise copy

• This can lead to unwanted sharing:
  stack stack1 (200);
  stack stack2 = stack1; // stack1.contents shared
  stack2.push (15); // stack1 is modified

• Need to redefine assignment and copy
Copy constructor

```cpp
stack (const stack& s) { // reference to existing object
    contents = new int [ sz = s.size()];
    for (int i = 0; i <sz; i++) contents [i] = s.contents [i];
}
stack s1 (100);
...
stack s2 = s1; // invokes copy constructor
```
Redefining assignment

• assignment can also be redefined to avoid unwanted sharing
• operator returns a reference, so it can be used efficiently in chained assignments:
  one = two = three;

  stack & operator=(const stack& s) {
    if (this != &s) {
      // beware of self-assignment
      delete [] contents;  // discard old value
      contents = new int [sz = s.size ()];
      for (int I = 0; I < sz; I++) contents [I] = s.contents [I];
    }
    return *this; }

  stack s1 (100), s2 (200); ...  s1 = s2; // transfer contents
Notes

• Just like with the compiler-given copy constructor, we can see that we might get into trouble with the compiler-given assignment behavior if one of our data members was a pointer!

• As such, we have to be vigilant and make sure that classes with pointer data members don't rely on the default assignment behavior.
Outline

• OOP revisit
• Debugging
• Dynamic memory allocation
Debugging

• Bug = error
• Debug = try to fix the error
  • Locate the bug (which lines of code introduce the bug)
  • Understand the bug
  • Fix it

• Debugger helps to locate and understand the bug
Outline

• OOP revisit
• Debugging
• Dynamic memory allocation
Static memory allocation

• If we want to type in a paragraph and save it into a C-string.

• #define MAXLENGTH 10000
  • char s[MAXLENGTH+1]; cin.getline(s);

• What if the paragraph is extremely long?
  • out-of-bound

• What if the paragraph has only five words?
  • over-allocated memory
Dynamic memory allocation

• What if we want to fit the paragraph into a C-string with right the sufficient size?

• Dynamic allocation of an array
  • `<type> *<name> = new <type>[<#elements>];`
  • `char *article = new char[length];`

```cpp
int length;
cout << how many characters are at most in your article? << endl;
cin >> length;
char *article;
if (length > 0)
    article = new char[length + 1];
```
New

• **new** will dynamically allocate the sequential memory space of the desired type and size.

• **new** will always return the **start address** of the allocated space.

• `int array = new int[size];` ✗
  `int *array = new int[size];` ✓
Memory leak

```c
int *p;
p = new int[2000000];
p = new int[1000000];
```

• We allocate 2,000,000 blocks of int and point p to it.
• Then we allocate another 1000000 and point p to it. p no longer points to the first 1000000 blocks.
• The first 2000000 blocks of int becomes a ghost. We can no longer access it and release it.
• This phenomenon is called Memory Leak.
Delete

• `delete[] p;`
• Delete the entire array pointed by p and release all the memory.
See you next week!