UCLA Samueli
Computer Science

CS31 Week 8 Discussion

Fall 2021, Section 1C
Mingyu Derek Ma mdma@ucla.edu
Thanks Muhao Chen and Rosa Garza for their shared content

https://derek.ma/cs31 for slides and other discussion materials

Reminder

- Project 6 due on the coming Monday, Nov 22nd
- Worksheet available for next week
- Next Monday office hours: Alexis will sub me for the first two hours

Project Suggestions

- Print out intermediate variable values to check whether they meet your expectation
- Use functions/loops to avoid repeating code for similar behavior
  - Repeated code can lead to typos and make the code more complicated and hard to debug
- Set global constant variable for magic numbers
  - It will be easier to understand and less duplication
- Check variable scope and array boundary
  - Do not define variable inside a function and use it outside the function
  - Make sure your loop/other statements do not use out-of-bound elements of an array
- Be aware of memory leak

struct

- To save records to some data structures, what can we do?
  - Saving the following information of students
    - Name
    - ID
    - Email
    - Grade

```
#define NUM_STUDENT 33
string name[NUM_STUDENT];
int id[NUM_STUDENT];
string email[NUM_STUDENT];
char grade[NUM_STUDENT];
```

- Using separate arrays to save different kind of information is inconvenient
  - What if we want to swap records of two students

Define a struct

```
struct student {
      string name;
      int id;
      string email;
      char grade;
}; //Note: there a semi-colon here
```

Declare objects of a struct

```
// declare an object
student eric;
// declare an array of objects
student students[33];
```

Initialize objects of a struct

```
struct student {
    string name;
    int id;
    string email;
    char grade;
};

student students[33];
students[0].name = "Eric";
students[0].id = 123456789;
students[0].email = "";
students[0].grade = 'C';
```

- Use `object_name.attribute_name` to access attribute
- Accessing attributes of an uninitialized struct object results in undefined behaviors

Access attributes of a struct pointer

```
struct student {
    string name;
    int id;
    string email;
    char grade;
};

student students[33];
students[0].name = "Eric";
students[0].id = 123456789;
students[0].email = "";
students[0].grade = 'C';

cout << students[0].name << endl;
student *s1;
s1 = &students[0];
cout << (*s1).name << endl;
cout << s1 -> name;
```

- We can use . with dereference
- More commonly: we can use –>
- Difference between . and –>
  - Left-hand side of . is a struct object
  - Left hand side of –> is a pointer to a struct object

Access attributes of a struct pointer

```
//Four ways to access attributes

student[0].name
s1 -> name

// following are not really used
in practice
(*s1).name
(&student[0])->name
```

- We can use . with dereference
- More commonly: we can use –>
- Difference between . and –>
  - Left-hand side of . is a struct object
  - Left hand side of –> is a pointer to a struct object

```
cout << s1.name << endl;
cout << *s1.name << endl;
```

Member reference type 'student *' is a pointer; did you mean to use '->'?

2 Indirection requires pointer operand ('std::string' (aka 'basic_string<char>') inval...

class

```
class vending_machine {
public:
        int get_num() const; //accessor
        double get_price() const; //accessor
        void set_num(const int& num);//modifier
private:
        int num;
        double price;

};
class human {
public:
        bool buy_one(const vending_machine &vm);
private:
        int num_items;
        double cash;
};
```

Member functions

```
// Accessor

int vending_machine::get_num() const {
        return num;             // style 1
        return this -> num;     // style 2
        // this is a pointer that points to the object itself
};

// Modifier

void vending_machine::set_num(const int& num){
        this -> num = num;
};
```

Constructors

- Functions to specify the behavior of object initiation
- Used to initialize member variables of the class/struct when we create an object of this class/struct
- Function name is the same as the class name, no return type specification
- Constructor without parameter:

```
vending_machine::vending_machine() {
        num = 10;
        price = 1.75;
};

vending_machine vm;
// vm is a vending machine object that sells 10 items at $1.75 each
```

Constructors

- Constructor with parameters

```
vending_machine::vending_machine(const int& num, const double &
price) {
        this->num=num;
        this->price=price;
};

vending_machine vm(30, 2.0); //vm sells 20 items at $2 each
```

Constructors

- If we do not specify any constructors for a class, an empty constructor will be provided by default without parameters. If we specify a constructor, the empty one will be overwritten

```
class human {
public:
        bool buy_one(const vending_machine &vm);
private:
        int num_items;
        double cash;
};
human::human(const int& num, const double & cash) {
        this->num_items = num;
        this->cash=cash;
};
```

- We cannot do `human hm;`
- Because the empty one is replaced by the specified one, so we should do `human hm(30, 80.5);`

Multiple Constructors with Different Parameter Types

```
class human {
public:
        bool buy_one(const vending_machine &vm);
private:
        int num_items;
        double cash;
};
human::human(const int& num, const double & cash) {
        this->num_items = num; this->cash=cash;
};
human::human(const double & cash) {
        this->num_items=0; this->cash=cash;
};
human::human() {
        this->num_items=0; this->cash=60.0;
};
```

- Corresponding constructor is called depending on the combination of parameter types when calling the constructor

Private member variables/functions

- A private member variable/function can only be seen by the code of this class
- Other classes, functions, main function cannot see private members

```
class vending_machine {
public:
        int get_num() const; //accessor
        double get_price() const; //accessor
        void set_num(const int& num);//modifier
private:
        int num;
        double price;
};
class human {
public:
        bool buy_one(const vending_machine &vm);
private:
        int num_items;
        double cash;
};
```

Private member variables/ functions

```
// Wrong implementation
bool human::buy_one(const &vending_machine vm) {
        if (vm.num <= 0 || this->cash <= vm.price)
                return false;
        vm.num -= 1;
        this->cash -= vm.price;
        return true;
}
```

```
// Right implementation
bool human::buy_one(const &vending_machine vm) {
        if (vm.get_num() <= 0 || this->cash <=
vm.get_price())
                return false;
        vm.set_num(vm.get_num() – 1);
        this->cash -= vm.get_price();
        return true;
}
```

Destructor

- Things to do when an object is destructed
- Will introduce on next Monday

Difference struct vs class

- They are the same besides their default member variable/function’s visibility
- struct: default set to public
- class: default set to private, more secure by default
- Operations are shared between struct and class

Dynamic Memory Allocation

- Static memory allocation is not flexible
  - If the data we want to save is too large, then it’s out-of-bound is we set the small limit
  - If the data we want to save is too small, then we waste a lot of memory
- Dynamic memory allocation
  - Allocate at runtime, not compile time

```
const int MAX_SIZE = 10000;
vending_machine vms_static[MAX_SIZE]; // static allocation
```

⚠ Unused variable 'vms_static'

```
// dynamic allocation
int vm_count = 0;
vending_machine * vms[MAX_SIZE];
vms[vm_count] = new vending_machine; // dynamically allocates an object
vm_count += 1;

// do this many times

// delete an object
delete vms[1];
for (int i = 1; i < vm_count - 1; i++)
    vms[i] = vms[i + 1]; // shift pointers forward
vm_count -= 1;

// once done using all vms, we need to manually delete those objects
for (int i = 0; i < vm_count; i++)
    delete vms[i];
```

Memory leak

- We need to manually delete the objects created by `new`, if not there is memory leak issue
- Need to make sure you keep the pointer point to a dynamically allocated object
  - vm does not point to the first object any more
  - We have no way to access or release it

```
vm = new vending_machine;
vm = new vending_machine;
delete vm;
```

Thank You