Week 9

Muhao Chen
muha ochen@ucla.edu
http://yellowstone.cs.ucla.edu/~muhao/
Outline

- Class
- Final Practice
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;
};
Implement simple member functions

Accessor:

```cpp
int vending_machine::get_num() const {
    return num;
};
```

Or we can say: (this is a pointer that points to the object itself).

```cpp
int vending_machine::get_num() const {
    return this -> num;
};
```

Modifier:

```cpp
void vending_machine::set_num(const int& num) {
    this -> num = num;
};
```
Constructors: functions to specify the behavior of object initiation.

Default constructor (no parameter):

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

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

Constructor with parameters:

```cpp
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`

Function name is the same as the class name. No return type specification.
Constructors: functions to specify the behavior of object initiation

If we do not specify any constructors for a class, an **empty constructor** will be provided by default. But it will not if we have specified a constructor.

```cpp
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;
};
```

**Can we do this?**

human hm;

**No.** We have to do:

human hm(0, 80.0);

Because default constructor is not available.
Constructors: functions to specify the behavior of object initiation

Multiple constructors with different combinations of 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;
};
Caution: private member variables/functions

A private member variable/function can only be seen by the code of this class.

```cpp
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;
};
```
Caution: private member variables/functions

To implement buy_one, can we do:

```cpp
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;
}

bool human::buy_one(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

```cpp
destructor: things to do when an object is destructed

vending_machine::~vending_machine() {
    cout<< "A vending machine is out of order."
};

int main() {
    vending_machine vm;
    return 0;
}
// we’ll see “A vending machine is out of order.”
```
Destructor

- A destructor is necessary when we have **dynamically allocated member variables**:

```cpp
class vending_machines {
    public:
        vending_machines(int num) {
            vms = new vending_machine*[num];
            for (int i=0; i<num; ++i)
                vms[i] = new vending_machine;
            this->num = num;
        }
    private:
        vending_machine **vms;
        int num;
};

vending_machines::~vending_machines() {
    for (int i=0; i<num; ++i) delete vms[i]
    delete[] vms;
}
```
Class: Things to be careful during the final (esp. coding parts)

- Do not access the private members of another class (look at page 9)
- Use the correct version of constructor (page 6)
- Release dynamic allocated items in the destructor (page 11)
Some practice for final
#1 True or False

<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
<th>Variable names can be started with an alphabet or a digit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
<td>A string cannot be empty.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>A character cannot be empty.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>An array can have a size of 0.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>int m = 5.6; won’t compile because types do not match.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>int n = 14 / 5; will create n and initialize it with the value of 2.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>An array index begins with 0.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>for (int i = 0; i &lt;= 49; i++) will run the loop 50 times (assuming i is not modified within the loop).</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>The C-string char s[100] can store at most 99 characters.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>Constant variables can be modified only in the main() function.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>int x = 0.0; sets x to an integer 0.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>int x = 0.5; sets x to an integer 0.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>double x = 3; sets x to a double float 3.0.</td>
</tr>
</tbody>
</table>
# Parameters

```c
int func1(int &a, int m, int &b) {
    m = a; a = b; b = m;
}
int main1() {
    int x=0, y=1;
    func1(x, NULL, y);
}

int func2(int &a, int &m, int &b) {
    m = a; a = b; b = m;
}
int main2() {
    int x=0, y=1;
    func2(x, NULL, y);
}

int func3(int *a, int *m, int *b) {
    int *t = a; a = b; b = t;
}
int main3() {
    int x=0, y=1;
    func3(&x, NULL, &y);
}

int func4(int *a, int *m, int *b) {
    int t = *a; *a = *b; *b = t;
}
int main4() {
    int x=0, y=1;
    func4(&x, NULL, &y);
}
```

Which functions swap x and y?

(A) main1  (B) main2  (C) main3  (D) main4

**G main2. We cannot pass NULL to a reference type.**

main3. it’s actually pass-by-value.
#3 Consider the following code

```cpp
string s1, s2("Hello");
cout << "Enter a line of input:\n";
cin >> s1;
if (s1 == s2)
    cout << "Equal\n";
else
    cout << "Not equal\n";
```

If our input on the screen is as below:

**Enter a line of input:**

Hello my friend! My name is Dimitri Petrenko.

What is the output?

(A) Not equal    (B) Equal  (C) Nothing

**B. cin flow terminates a single >> with a whitespace or Enter.**
#4 Assume the following variable declarations:

```c
int foo = 0;
int *ptr = &foo;
```

Which of the following statements will change the value of foo to 1?

(A) `*ptr++`;  (B) `foo++`;  (C) `(*foo)++`;  (D) `(*ptr)++`;
(E) A & B & D  (F) A & D  (G) B & D  (H) C & D

G  ++ suffix has higher precedence than *.
#5 Consider following four functions

```c
void s_toLowerA(char s[]) {
    int i=0;
    while (s[i] != '\0')
        s[i++] = tolower(s[i]);
}

void s_toLowerB(char s[]) {
    int i=0;
    while (s[i] != '\0')
        s[i] = tolower(s[i++]);
}

void s_toLowerC(char s[]) {
    int i=0;
    while (s[i] != '\0')
        s[++i] = tolower(s[i]);
}

void s_toLowerD(char s[]) {
    int i=0;
    while (s[i] != '\0')
        s[i] = tolower(s[i]); ++i;
}
```

Which of them correctly changes any C-string to its lowercase?
- (A) s_toLowerA
- (B) s_toLowerB
- (C) s_toLowerC
- (D) s_toLowerD
- (E) A & C & D
- (F) A & B & D
- (G) A & B & C & D
- (H) A & D

**F** `++` suffix has lower priority than `=` and parameter passing.
We know that `i=j++` assigns `j` to `i` before increases `j`. Even `++` is a part of right-hand side parameter, it still increases after assignment.
Which of the following changes kitty1’s age into 5?
(A) kitty1.age = 5;  (B) (&kitty1)->age = 5;  (C) kitty1.getAge() = 5;
(D) kitty1.setAge(5);  (E) C & D  (F) A & B  (G) A & B & C & D  (H) A & B & D

D. A member without indicating “private” or “public” is by default private in a class.
   - getAge() only returns a read-only (immutable) value of age.
#7 Modify the age of the cat

```cpp
struct cat{
    int age;
    public:
        cat(){};
        ~cat{};
        void getAge() { return age; };
        void setAge(int n) { this -> age = n;};
};
cat kitty1 = *(new cat());
```

Which of the following changes kitty1’s age into 5?
(A) kitty1.age = 5;  (B) (&kitty1)->age = 5;  (C) kitty1.getAge() = 5;
(D) kitty1.setAge(5);  (E) C & D  (F) A & B  (G) A & B & C & D  (H) A & B & D

**H.** The only difference between struct and class in C++: if you do not indicate whether a member is public or private, then by default: **private in class; public in struct;**

**Note:** for C, even if age is now public, getAge() is still immutable
#8 Write delete statements to delete all dynamically allocated memory

```cpp
define p1 = new int[10];
define p2[15];
for (int i = 0; i < 15; i++)
  p2[i] = new int[5];

int **p3 = new int*[5];
for (int i = 0; i < 5; i++)
  p3[i] = new int[7];

int *p4 = new int;
define temp = p4;
p4 = p1;
p1 = temp;

delete p1;  // p1 points to the int previously allocated to p4

for (int i = 0; i < 15; ++i)
  delete[] p2[i];  // delete each row of p2

for (int i = 0; i < 5; ++i)
  delete[] p3[i]; // delete each row of p3

delete[] p3;  // then delete the entire p3

delete[] p4;
```
#9 Design a data stream window

- A data stream is a system where data points come one after another. For example, we may use a data stream window to record the stock price within a period of time.

- At each time we can only use a $k$-sized “sliding window” to store $k$ newest data. $k$ is fixed. Each time we receive a data, we insert it into the window.

- If the window has already been filled up with $k$ data points, the oldest one will expire (i.e. remove from the window) when we insert the new data.
#9 Design a data stream window

- This window supports following functionalities:
  1. `window(int k)`: constructor, create a window which can store up to $k$ data.
  2. `receive(double newdata)`: we receive a new data and add into the window. If the sliding window has already got $k$ data before we insert the new data, the oldest data will expire.
  3. `average()`: return the current average of all data in the window.
  4. `min(), max()`: return the current min/max data in the window.
  5. `num()`: return # of data currently stored in the window
  6. `size()`: return the size of the window (i.e. $k$)
  7. `[i]`: access the i-th data in the window. Return the closest element if i exceeds boundary. (Extra point)
#9 Design a data stream window

//How we may want to use this window.
window win = *(new window(3)); //the window can store 3 data
win.receive(10.0); //received first data {10.0}
cout << win.num(); //we get 1
cout << win.size(); //we get 3
win.receive(15.0); //received second data {10.0, 15.0}
cout << win.average(); // we get 12.5
win.receive(20.0); //received third data {10.0, 15.0, 20.0}
win.receive(25.0); //received fourth data, the oldest data 10.0 expires
                 // {15.0, 20.0, 25.0}
cout << win.min(); // we get 15.0
win.receive(13.0); // {20.0, 25.0; 13.3}
cout << win.max(); // we get 25.0

cout << win[2]; // we get 13.3
Now implement the class `window`

class window {
public:
    window(int n) {
        k = n; avg = num = 0;
        data = new double[n]; // a k-size double array to store k data
    }
    ~window() { delete[] data; }
    void receive(double newdata);
    double average() const;
    double max() const;
    double min() const;
    int size() const;
    int num() const;
    double &operator[ ] (int i);
}

private:
    int k, num; // size, and current # of data points
    double *data; // pointer to the beginning of the sliding window
    double avg;
}
```cpp
double window::receive(double newdata) {
    if (num < k) { // window is not filled
        data[num++] = newdata;
        avg = (avg * (num - 1) + newdata) / num; //online maintaining average
    }
    else { //if window is filled, expire the oldest
        avg = avg - data[0] / k + newdata / k; //update avg as one expires and another is inserted
        for (int i=0; i < k - 1; ++i) data[i] = data[i + 1]; //move left data[2..k-1] by 1
        data[k - 1] = newdata; //insert new data;
    }
}
```
double window::average() const {
    return this->avg;
}

double window::min() const {
    double m = DBL_MAX;
    for (int i=0; i<k; ++i) if (data[i] < m) m = data[i];
    return m;
}

double window::max() const {
    double m = DBL_MIN;
    for (int i=0; i<k; ++i) if (data[i] < m) m = data[i];
    return m;
}

int window::size() const {
    return k;
}

int window::num() const {
    return num;
}
double& window::operator[](int i) {
    if (i < 0) return data[0];
    if (i >= num) return data[num-1];
    return data[i];
}
#10 Remote control typing

Suppose we are using a remote control to type words on TV. We have a virtual keyboard on the screen (e.g. a keyboard of width 9 as below). A cursor is placed on the position of. We can either move the cursor to four directions with \(^v<\), or select the letter with \(*\) (selection will not restore the position of the cursor to A).

\[
\begin{array}{cccccccc}
A & B & C & D & E & F & G & H & I \\
J & K & L & M & N & O & P & Q & R \\
S & T & U & V & W & X & Y & Z \\
\end{array}
\]

To type “CAT”, we need to type “>>>*<<*>*VV*”. Now write a function, given a word and the width of the keyboard (i.g. # of letters in each row of the keyboard), convert the word to its “remote control encoding”.

```cpp
string encode(string word, int width)
```
Solution

- Divide the problem into three steps:
  1. Given a letter, what is its position (row, col)?
  2. Given a source position and a target position, what’s the “remote control” encoding to move from source to target.
  3. For a word, concat all the “remote control version” strings for each chars.
position of letter, movement between positions

```cpp
void position(char letter, int width, int &row, int &col) {
    int gap = letter - 'A';
    row = gap / width;
    col = gap - row * width;
}

string move(int row_t, int col_t, int row_s, int col_s) {
    string res = "";
    if (row_t > row_s) //move downwards
        for (int i=0; i<row_t - row_s; ++i)res += "v";
    else if (row_t < row_s) //move upwards
        for (int i=0; i<row_s - row_t; ++i)res += "^";
    if (col_t > col_s) //move >
        for (int i=0; i<col_t - col_s; ++i)res += ">";
    else if (col_t < col_s) //move <
        for (int i=0; i<col_s - col_t; ++i)res += "<";
    return res + "*";
}
```
string encode(string word, int width) {
    string res="";
    int row_s, col_s, row_t, col_t;
    row_s = col_s = 0; //position of A
    for (int i=0; word[i] != '\0'; ++i) {
        position(word[i], width, row_t, col_t);
        res += move(row_t, col_t, row_s, col_s);
        row_s = row_t; //current target position is the source position for next char
        col_s = col_t;
    }
    return res;
}
Thank you. Best of luck.