Queens College, CUNY, Department of Computer Science

Object-Oriented Programming in C++

CSCI 211/611

Summer 2018

Instructor: Dr. Sateesh Mane

due date Friday, July 20, 2018, 11.59 pm

Homework: Classes: functions and methods

- **Experience with other classes has demonstrated that in many cases the source of difficulty is not the mathematics or the programming.**
- **The source of difficulty is the English (understanding the text).**
- **If you do not understand the words in the lectures or homework, THEN ASK.**
- **If you do not understand the concepts in the lectures or homework, THEN ASK.**
- **Send me an email, explain what you do not understand.**
- **Do not just keep quiet and then produce nonsense in exams.**

- **Consult your lab instructor for assistance.**
- **You may also contact me directly, but I cannot promise a prompt response.**
- Please submit your inquiry via email, as a file attachment, to `Sateesh.Mane@qc.cuny.edu`.
- Please submit one zip archive with all your files in it.
  1. The zip archive should have either of the names (CS211 or CS611):

     `StudentId_first_last_CS211_hw_classes1.zip`
     `StudentId_first_last_CS611_hw_classes1.zip`
  2. The archive should contain one "text file" named "hw_classes1.[txt/docx/pdf]" (if necessary) and cpp files named "Parent_child.cpp" and "Student.cpp" etc.
  3. Note that a text file is not always required for every homework assignment.
  4. Note that not all questions may require a cpp file.

General information

- You should include the following header files, to run the programs below.

  ```
  #include <iostream>
  #include <iomanip>
  #include <string>
  #include <vector>
  #include <cmath>
  ```

- If you require additional header files to do your work, feel free to include them.
- **Include the list of all header files you use, in your solution for each question.**
- The questions below do not require complicated mathematical calculations.
- If for any reason you require help with mathematical calculations, **ask the lab instructor or the lecturer.**

Q1 Classes `Parent` and `Child`

- We shall write two classes `Parent` and `Child`.
- The `Parent` class has a vector of `Child` objects.
- The `Child` class has a `Parent` object.
- *Hence the classes refer to each other.*
- **Therefore we must write forward declarations for both classes.**
- We shall do so in this homework assignment.

Q2 Forward declaration

- **Write the following forward class declarations for `Parent` and `Child`.**

```
class Child;                          // forward statement that class "Child" exists
class Parent;                         // forward statement that class "Parent" exists

class Parent {
public:
  Parent();

  string getName() const;
  void setName(string n);
  void addChild(const Child &c);
  int numChildren() const;
  const vector<Child>& getChildren() const;

private:
  string name;
  vector<Child> children;
};

class Child {
public:
  Child(string n, int a, const Parent &pt);

  string getName() const;
  int getAge() const;
  const Parent* getParent() const;

private:
  string name;
  int age;
  Parent p;
};
```

- In the `Parent` class, the reference to the vector for `getChildren()` is `const` because we do not want an external application to change somebody's children.
- In the `Child` class, the pointer for `getParent()` is `const` because we do not want an external application to change somebody's parent.

Q3 Class `Parent` function definitions

- **Write the following code below both class declarations.**
- Write non-inline function definitions for the class `Parent`.
- **Write "`Parent::`" in front of all the function names, including the constructor.**
- The constructor is empty. There is nothing to do.

```
Parent::Parent() {}                              // empty constructor
```

- The class methods do something. Some are `const`. Fill in the function bodies.

```
string Parent::getName() const                   // return name

void Parent::setName(string n)                   // set name

void Parent::addChild(const Child &c)            // push back "c" onto vector

int Parent::numChildren() const                  // return size of vector

const vector<Child>& Parent::getChildren() const // const method
                                                 // return const reference to vector
```

Q4 Class Child function definitions

- **Write the following code below both class declarations.**
- Write non-inline function definitions for the class Child.
- *Note: Because both forward class declarations have been written, it does not matter if the non-inline code for* Child *is written before or after the non-inline code for* Parent.
- **Write "Child::" in front of all the function names, including the constructor.**
- The constructor for Child is a non-default constructor.

```
Child::Child(string n, int a, const Parent &pt)
{
  // set values of data members using the inputs
}
```

- The class methods do something. All are const. Fill in the function bodies.

```
string Child::getName() const                   // return name

int Child::getAge() const                       // return age

const Parent* Child::getParent() const          // const method
                                                // return const pointer
```

Q5 Main program for forward declarations

- Test your code with the following main program.
- There are two parents Alice and Bob.
- Charlie and Dora (ages 5 and 6) are children of Alice and Elizabeth (age 7) is a child of Bob.

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

// forward class declarations and non-inline code for function bodies

int main()
{
  Parent a, b;
  a.setName("Alice");
  b.setName("Bob");

  Child c("Charlie", 5, a);
  Child d("Dora", 6, a);
  Child e("Elizabeth", 7, b);

  a.addChild(c);
  a.addChild(d);
  b.addChild(e);

  cout << "Parent of " << c.getName() << " is " << c.getParent()->getName() << endl;
  cout << "Parent of " << d.getName() << " is " << d.getParent()->getName() << endl;
  cout << "Parent of " << e.getName() << " is " << e.getParent()->getName() << endl;

  cout << endl;
  cout << "Children of " << a.getName() << endl;
  const vector<Child> &av = a.getChildren();
  for (int i = 0; i < av.size(); ++i)
    cout << setw(12) << av[i].getName() << setw(12) << av[i].getAge() << endl;

  cout << endl;
  cout << "Children of " << b.getName() << endl;
  const vector<Child> &bv = b.getChildren();
  for (int i = 0; i < bv.size(); ++i)
    cout << setw(12) << bv[i].getName() << setw(12) << bv[i].getAge() << endl;

  return 0;
}
```

Q6 Classes `Student`

- **Write a class `Student` with private data and public methods.**

- The class has two private data members of type `string` and `vector<double>`.

```
class Student {
private:
   string name;
   vector<double> grades;

public:
  // to do
};
```

- We shall write additional class methods, to be described below.

- Some of the methods will be tagged `const`.

Q7 Accessors and mutators

- **Write public accessor and mutator methods.**

```
string getName() {...}                        //should be "const"
void setName(const string &s) {...}
string&  nameRef() {...}
```

- The method `nameRef()` is safe, because the reference is to the data member `name`, and `name` does not go out of scope at the function exit.
- Because the return value is a reference to `name`, **the method `nameRef()` is both an accessor and a mutator.**

```
st.nameRef() = "something";           // method is mutator
cout << st.nameRef() << endl;         // method is accessor
```

- Because `nameRef()` can be employed as a mutator, it is not `const`.
- **Write a mutator to add a grade to the vector `grades`.**

```
void addGrade(double x) {...}
```

1. If $x \geq 0$ and $x \leq 100$, populate the vector `grades` with the value x.
2. Else return and do nothing.

Q8 `const` methods

- **Write a method to calculate and return the average grade.**
- If the size of `grades` is zero, then return 0.

```
double getAvg() // etc
```

- **Write a method to return the highest grade.**
- If the size of `grades` is zero, then **return** -1**.**

```
double highestGrade() // etc
```

- **Write a method to print() the name and grades.**

```
void print() // etc
```

1. First print `name`.
2. Next print the grades in a loop, one grade on each line.
3. Print a message “no grades posted yet” if the size of `grades` is zero.

- **Explain why all of the methods in this secton are `const`.**

Q9 `non-const` methods

- **Write a method to return a pointer to the address of an element of `grades`.**

```
double* gradePtr(int n) {...}
```

- Return the address of `grades[n]` if the value of n is valid.
- Else return NULL.
- **Explain why this method is not `const`.**

Q10 Class declaration

- **Your overall class declaration should look like the following.**

```
class Student {
private:
   string name;
   vector<double> grades;

public:
  string getName();                     // apply keyword "const" correctly
  void setName(const string &s);
  string& nameRef();

  void addGrade(double x);

  // non-const methods
  double * gradePtr(int n);

  // const methods
  double getAvg();                      // apply keyword "const" correctly
  double highestGrade();                // apply keyword "const" correctly
  void print();                         // apply keyword "const" correctly
};
```

Q11 Functions

- **Write two functions as follows to use your code.**

  ```
  void highlow_avg_grade(Student *a, Student *b, int n);
  void highlow_top_grade(Student *a, Student *b, int n);
  ```

- In both functions, *a* is a pointer to a single object and *b* is a pointer to an array of length *n*.
- First function:
  1. Find the name and average grade of the student with the highest average grade.
  2. Find the name and average grade of the student with the lowest average grade.
  3. **Print output to screen.**

     ```
     cout << "high avg = " << name_high << "  " << high << endl;
     cout << "low avg  = " << name_low << "  " << low << endl;
     ```

- Second function:
  1. Same as the first function but replace `getAvg()` by `highestGrade()`.
  2. **Print output to screen.**

     ```
     cout << "high top grade = " << name_high << "  " << high << endl;
     cout << "low top grade  = " << name_low << "  " << low << endl;
     ```

- **For both functions, state if the inputs can be tagged as “const” pointers.**
- **If yes, then change the function signature to declare them as `const` pointers.**

Q12 Example main program

- **Your code should work correctly when tested with the following main program.**

```
// include headers, class declaration, functions

int main()
{
  Student *Alice = new Student;
  Student *BobTwins = new Student[2];

  // use nameRef() to set name of Alice to "Alice";
  // use setName(...) to set names of BobTwins to "Bob A" and "Bob B"

  // call print() for Alice and BobTwins

  for (int i = 65; i <= 110; i += 10) {
    //  addGrade(i+0.1)    add grades for Alice
  }

  for (int i = 57; i <= 110; i += 10) {
    // addGrade(i+0.2)     add grades for BobTwins[0]
  }

  int igrade=0;
  while (true) {
    // double *d = ... gradePtr(igrade)    pointer to double for BobTwins[0]
    // if d == NULL then break out of loop
    // addGrade(*d - 0.5)     add grades for BobTwins[1]

    ++igrade;                 // increment counter
  }

  // call print() for Alice and BobTwins

  highlow_avg_grade(..., ..., 2);       // call functions
  highlow_top_grade(..., ..., 2);

  // release memory as appropriate

  return 0;
}
```