Queens College, CUNY, Department of Computer Science
Object-Oriented Programming in C++
CSCI 211/611
Summer 2018
Instructor: Dr. Sateesh Mane

due date Monday, July 23, 2018, 11.59 pm

Homework: operator overloading

- **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_op_overload.zip`
     `StudentId_first_last_CS611_hw_op_overload.zip`
  2. The archive should contain one "text file" named "hw_overload.[txt/docx/pdf]" (if necessary) and one cpp file per question named "Q1.cpp" and "Q2.cpp" etc.
  3. Note that not all questions may require a cpp file.
  4. A text file is not always required for every homework assignment.

General information

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

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

Q1.1 Previous class declaration

- Recall the declaration of the class Vec_int.
- **We shall overload some operators for the class Vec_int.**

```
class Vec_int {
public:
  Vec_int();
  Vec_int(int n);
  Vec_int(int n, int a);

  Vec_int(const Vec_int &orig);
  Vec_int& operator= (const Vec_int &rhs);
  ~Vec_int();

  int capacity() const;
  int size() const;

  int front() const;
  int back() const;

  void clear();
  void pop_back();
  void push_back(int a);

  int& at(int n);

private:
  void allocate();
  void release();

  int _capacity;
  int _size;
  int * _vec;
};
```

Q1.2 `operator []`

- The `Vec_int` class has a method `at(int n)`.
- We can write code such as this:

```
Vec_int v;
...
int i = 3;
int j = v.at(0);
v.at(i) = 7;
```

- However we would like to write simpler expressions such as this:

```
Vec_int v;
...
int i = 3;
int j = v[0];
v[i] = 7;
```

- **Declare two operators as public methods of `Vec_int`.**

```
int& operator[] (int n);
const int& operator[] (int n) const;
```

1. They are both public.
2. The reason we require two versions of `operator[]` is so that `const` objects can invoke the second version.
3. That is why the return type of the second version is a **const reference.**

- The non-const version of `operator[]` is easy. Just return `at(n)`.

```
int& Vec_int::operator[] (int n)
{
  return at(n);
}
```

- The const version of `operator[]` is actually also easy.

1. However, we cannot call `at(n)` because `at(int n)` is not a `const` method.
2. Instead we copy and paste the code of `at(int n)`.

```
const int& Vec_int::operator[] (int n) const
{
  // copy and paste code of Vec_int::at(int n)
}
```

3. The only difference between the functions is that the return type is `const int&`.

Q1.3 `operator +`

- The `string` class allows us to add two `string` objects.

```
string s1, s2, s3;
...
s1 = s2 + s3;
```

- The value of `s1` is the concatenaton of `s2` and `s3`.
- **We shall overload `operator+` to concatenate two `Vec_int` objects.**
- **Declare an overloaded `operator+` with the following signature.**

```
Vec_int operator+ (const Vec_int &u, const Vec_int &v);
```

- **This is not a class method. This is an external function (actually, an operator).**
- Write the function body of the operator.
- There are multiple ways to write the code.
  1. Declare a local variable `Vec_int w` inside the function body.
  2. Let the sizes of u and v be s_u and s_v respectively.
  3. The data values in w should be the following:
  $$u[0], \dots, u[s_u - 1], v[0], \dots, v[s_v - 1]\,.$$
  4. The size of w should be $s_u + s_v$.
  5. At the end, return w.
- **The operator must work even if u and/or v are empty.**
- All correct implementations will be accepted, but they must not make use of the `vector<int>` class.
- Create two vectors u and v, populate them with data and print the data in $u + v$, also $v + u$.

Q1.4 operator <<

- It would be nice to print the contents of a Vec_int object by writing code like this.

```
Vec_int v;
...
cout << v << endl;
```

- **To do this we must overload operator <<.**
- However, to do so we require the ostream class ("output stream"), which may not be familiar to you.
- Here is the function code.

```
ostream& operator<< (ostream &os, const Vec_int &v)
{
  os << "( ";
  for (int i = 0; i < v.size(); ++i) {
    os << v[i] << " ";
  }
  os << ")";
  return os;
}
```

- **The const version of operator [] is invoked because v is const.**
- Create a Vec_int object v, populate it with some data, and execute the statement cout << v << endl; and see what it prints.

Summary

- The overall the declaration of the class `Vec_int` and the functions looks like this.

```
class Vec_int {
public:
  Vec_int();
  Vec_int(int n);
  Vec_int(int n, int a);

  Vec_int(const Vec_int &orig);
  Vec_int& operator= (const Vec_int &rhs);
  ~Vec_int();

  int capacity() const;
  int size() const;

  int front() const;
  int back() const;

  void clear();
  void pop_back();
  void push_back(int a);

  int& at(int n);

  int& operator[] (int n);                      // operator []
  const int& operator[] (int n) const;          // const version

private:
  void allocate();
  void release();

  int _capacity;
  int _size;
  int * _vec;
};


Vec_int operator+ (const Vec_int &u, const Vec_int &v);  // overload operator+

ostream& operator<< (ostream &os, const Vec_int &v);     // overload operator<<
```