Queens College, CUNY, Department of Computer Science

Object-oriented programming in C++

CSCI 211 / 611

Summer 2018

Instructor: Dr. Sateesh Mane

Project 1

due date Sunday July 8, 2018, 12:00 noon

- **NOTE: It is the policy of the Computer Science Department to issue a failing grade to any student who either gives or receives help on any test. A student caught cheating on any question in an exam, project or quiz will fail the entire course.**
- **Students who form teams to collaborate on projects *must inform the lecturer of the names of all team members* ahead of time, else the submissions will be classified as cheating and will receive a failing grade.**
- **Any problem to which you give two or more (different) answers receives the grade of zero automatically.**
- Please submit your solution 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_project1_July2018.zip`
     `StudentId_first_last_CS611_project1_July2018.zip`
  2. The archive should contain one "text file" named "Project1.[txt/docx/pdf]" and one cpp file per question named "Q1.cpp" and "Q2.cpp" etc.
  3. Note that not all questions may require a cpp file.
- **In all questions where you are asked to submit programming code, programs which display any of the following behaviors will receive an automatic F:**
  1. Programs which do not compile successfully (non-fatal compiler warnings are excluded).
  2. Array out of bounds, reading of uninitialized variables (including null pointers).
  3. Operations which yield NAN or infinity, e.g. divide by zero, square root of negative number, etc. *Infinite loops.*
  4. Programs which do NOT implement the public interface stated in the question.
- **In addition, note the following:**
  1. All debugging statements (for your personal testing) should be commented out.
  2. Program performance will be graded solely on the public interface stated in the questions.

Students who did not take Midterm 1 (in class) are not eligible for credit for this project.

Students who took the makeup Midterm 1 (take home) are not eligible for credit for this project.

General information

- **Ignore the statements below in this section if they are not relevant for the questions in this document.**
- **You are permitted to copy and use code supplied in the online lecture notes.**
- **64–bit computers**

  1. Hardware architecture will not be taken into account in the questions below.
  2. Solutions involving the writing of programs will not be judged if they work (or were written) on a 64–bit instead of a 32–bit computer.

- **The following refers to questions involving mathematical calculations.**

  1. **Value of π to machine precision on any computer.**
  2. Some compilers support the constant `M_PI` for π, in which case you can write

     ```
     const double pi = M_PI;
     ```

  3. If your compiler does not support `M_PI`, the value of π can be computed via

     ```
     const double pi = 4.0*atan2(1.0,1.0);
     ```

1 Material to be used in later questions

- **This is material for use in other questions. Nothing to "calculate" here.**
- Form a set of eight digits $(d_1, \ldots, d_8)$ as follows.
- **Take the 8 digits of your student id and define $(d_1, \ldots, d_8)$ as follows:**

$$\begin{aligned} d_1 &= \text{digit 1 of student id}\,, \\ d_2 &= \text{digit 2 of student id}\,, \\ d_3 &= \text{digit 3 of student id}\,, \\ d_4 &= \text{digit 4 of student id}\,, \\ d_5 &= \text{digit 5 of student id}\,, \\ d_6 &= \text{digit 6 of student id}\,, \\ d_7 &= \text{digit 7 of student id}\,, \\ d_8 &= \text{digit 8 of student id}\,. \end{aligned} \tag{1.1}$$

- For example if your student id is 23054611, then

$$\begin{aligned} d_1 &= 2\,, \\ d_2 &= 3\,, \\ d_3 &= 0\,, \\ d_4 &= 5\,, \\ d_5 &= 4\,, \\ d_6 &= 6\,, \\ d_7 &= 1\,, \\ d_8 &= 1\,. \end{aligned} \tag{1.2}$$

- *For some student ids, it is possible that some of the digits may be zero. It is also possible that some of the digits may be equal.* Do not worry.
- For the student id 11111111, all the digits are equal.
- For the student id 33330000, four digits are zero and the other four are all equal to 3.

2 Question 2

- **You will receive zero (or reduced) credit for this question if you did not answer Question 3 in Midterm 1.**
- **Consult the definition of the bounding rectangle from Midterm 1.**
- The definition is repeated below.
- Let (x_j, y_j), $j = 0, \ldots, n-1$ be an arbitrary set of n integer-valued coordinates.
- Hence the values of x_j and y_j are integers.
- In this question, we define the **bounding rectangle** as follows.
  1. The rectangle has horizontal and vertical edges.
  2. It is the **smallest rectangle** which encloses all the points (x_j, y_j), $j = 0, \ldots, n-1$.
  3. Let the coordinates of the bounding rectangle be (u_0, v_0), (u_1, v_1), (u_2, v_2) and (u_3, v_3).

$$\begin{array}{lcl} (u_0, v_0) & = & \text{bottom left corner} \\ (u_1, v_1) & = & \text{bottom right corner} \\ (u_2, v_2) & = & \text{top right corner} \\ (u_3, v_3) & = & \text{top left corner} \end{array}$$

- **Write a function to compute the bounding rectangle of the points (x_j, y_j), $j = 0, \ldots, n-1$.**
- The function signature is given below. Note that the return type is `int`.

```
int bounding_rect(int n, int x[], int y[], int u[], int v[]);
```

- **The return value is the area of the bounding rectangle.**
  The formula for the area A is as follows:

$$A = (u_2 - u_0)(v_2 - v_0)\,.$$

- **Submit your function code as your solution.**
  1. Write the function only.
  2. Do not write a main program.
  3. Do not worry about header files (for math, etc.). Assume they are all given to you.

3 Question 3

- You are given an input array a of type `int` and length n.
- You are also given that the array **has no duplicates,** i.e. $a_i \neq a_j$ if $i \neq j$.
- **Write a function to sum the m largest elements in an array of length n.**
  1. The function signature is:

     ```
     bool sum_top(int a[], int n, int m, int & sum);
     ```

  2. The function return type is `bool`.
  3. **Safeguard:** if $n \leq 0$ or $m \leq 0$ or $m > n$ then set `sum = 0` and **return false.**
  4. If the calculation is sucessful, then **return `true`.**
  5. The value of the sum is contained in the output variable `sum`.
- There are many ways to implement `sum_top(...)`.
- **All correct solutions will be accepted.**
- If you know how to sort an array, you may sort the array a.
  1. Then sum the m largest elements in the sorted array.
  2. This solution will be accepted.
  3. **However you must write the sort function yourself.**
  4. **You must also explain the algorithm of the sort function.**
  5. Calling a 'black box' library function to perform the sort **will receive a failing grade.**

- **See next page.**

- Here is another possible implementation, which you may use.
  It is not necessarily the simplest or the fastest.
  It also has the bad feature that it changes the input array a (it is a partial sort).

  1. First implement the safeguard checks described above.
  2. Do not calculate anything if the safeguard checks fail (set `sum=0` and `return false`).
  3. Find the index of the largest element in the array a, call it say `imax`.
  4. Swap `a[n-1]` and `a[imax]`. Then `a[n-1]` contains the largest element of a.
     (a) Repeat the procedure as many times as required
     (b) Next find the index of the largest element in *the first $n-1$ elements of a.*
     (c) Swap `a[n-2]` and the element that you found. Then `a[n-2]` contains the second largest element of a.
     (d) Then find the index of the largest element in *the first $n-2$ elements of a.*
     (e) Swap `a[n-3]` and the element that you found. Then `a[n-3]` contains the third largest element of a.
  5. Stop the procedure when the m largest elements have been found.
  6. If you do your work correctly, the m largest elements of a will be swapped into the locations `a[n-1], a[n-2], ..., a[n-m]`.
  7. The required value is `sum` $=$ `a[n` $-$ `1]` $+$ `a[n` $-$ `2]` $+ \cdots +$ `a[n` $-$ `m]`.
  8. Return `true` and exit the function.

- The above implementation works, but has the bad side effect that it changes the array a.
- The calling application may not like that.
- We were asked to sum the m largest elements of a, *not* to modify the contents of a.
- The calling application might wish to use the array for other calculations, and we have disturbed the array.
- **Bonus:**
  Write your implementation such that it does not change the array a.

4 Question 4

- **Write a function to calculate the power of a number x^n for a double x and integer n, using the "divide and conquer" algorithm.**

  ```
  double power(double x, int n);
  ```

- The "divide and conquer" algorithm is explained below.

  1. **Safeguard: if $x == 0$ then return 0.** Don't waste time on technicalities with zero.
  2. If $n < 0$ then return `power(1.0/x, -n)`.
  3. Next, if $n == 0$ then return 1.0.
  4. Otherwise note the following (this is the "divide and conquer").
  5. The power function (e.g. math library function "`pow(x,n)`") has the following property.

     ```
     pow(x, n) =     pow(x, n/2) * pow(x, n/2)     // n is even
     pow(x, n) = x * pow(x, n/2) * pow(x, n/2)     // n is odd
     ```

  6. **Therefore the calculation of the powers of x can be implemented recursively.**

- **Write the function `power(double x, int n)` and implement the "divide and conquer" algorithm.**
- **Submit your function code as your answer.**
- You need not submit a main program or header files (for math, etc.).

5 Question 5

- This was the second last question in Midterm 1. It is possible that you ran out of time and did not answer the question in Midterm 1.
- **You will receive credit for submitting an answer to this question even if you did not answer the corresponding question in Midterm 1.**
- The **Babylonian algorithm** to calculate the square root of a positive number was described in class and is as follows.

  1. Let the number be a (you are given that $a > 0$).
  2. Let $x_0 = 1$.
  3. Compute successive iterates x_i, $i = 1, 2, \ldots$ via the following rule.

$$x_i = \frac{1}{2}\left(x_{i-1} + \frac{a}{x_{i-1}}\right).$$

  4. The iteration clearly goes on forever, so we set a tolerance to stop the iteration.
  5. **Stop the iteration when $|x_i - x_{i-1}| \leq 10^{-4}$.**
  6. **Return the value of x_i as the approximation to the square root $\sqrt{a}$.**

- **Write a function to implement the Babylonian algorithm to calculate the square root of a positive number.**

```
double sqrt_Babylonian(double a);
```

- The function return value is the approximation x_i to the square root of a.
- **Print the values of the iterates x_i inside your function, starting with $x_0 = 1$.**
- **Submit your function code as your answer.**
- You need not submit a main program or header files (for math, etc.).
- **Use your function to compute the square root of your student id.**
- Fill the table below with your list of iterates. State your values to 4 decimal places.

i	x_i
0	1
1	4 d.p.
2	4 d.p.
etc.	

6 Question 6: Mandatory for graduate students (bonus for undergraduates)

- **Write the body of a function with the following signature.**

```
bool divisible_9(int n, int &s);
```

- You are given an input positive integer $n > 0$.
- The decimal digits of n are $x_1 x_2 x_3 \ldots$.
- The sum of the digits $s = x_1 + x_2 + \ldots$.
- **If $s \leq 9$ then return `true` if $s == 9$, else return `false`.**
- **If $s > 9$ then set $n = s$ and call the function recursively until you obtain $s \leq 9$.**
  1. For example if $n = 999$ then proceed recursively until you obtain $s = 9$ and return `true`.
  2. For example if $n = 9876$ then proceed recursively until you obtain $s = 3$ and return `false`.
- *If you do your work correctly, the function will return* `true` *if n is divisible by 9 (and the output will be $s = 9$) and* `false` *if not (and the output will be a value $1 \leq s \leq 8$).*
- **Apply your function to your student id and state the output.**

7 Question 7: Mandatory for graduate students (bonus for undergraduates)

- **Write the body of a function with the following signature.**

```
bool divisible_11(int n, int &s);
```

- You are given an input positive integer $n > 0$.
- The decimal digits of n are $x_1 x_2 x_3 \ldots$.
- The sum of the digits in the odd locations is $s_1 = x_1 + x_3 + \ldots$.
- The sum of the digits in the even locations is $s_2 = x_2 + x_4 + \ldots$.
- **Set s to the absolute difference $s = |s_1 - s_2|$.**
- **If $s == 0$ then return `true`.**
- **Else if $s < 11$ return `false`.**
- **If $s \geq 11$ then set $n = s$ and call the function recursively until you obtain $0 \leq s < 11$.**
  1. For example if $n = 65432$ then $s_1 = 6 + 4 + 2 = 12$ and $s_2 = 5 + 3 = 8$. The absolute difference is $|s_1 - s_2| = 12 - 8 = 4$ hence return `false`.
  2. For example if $n = 7909$ then $s_1 = 7 + 0 = 7$ and $s_2 = 9 + 9 = 18$. The absolute difference is $|s_1 - s_2| = |7 - 18| = 11$ hence proceed recursively. The new values are $s_1 = 1$ and $s_2 = 1$, hence $|s_1 - s_2| = |1 - 1| = 0$ hence return `true`.
- *If you do your work correctly, the function will return* `true` *if n is divisible by 11 (and the output will be $s = 0$) and* `false` *if not (and the output will be a value $1 \leq s \leq 10$).*
- **Apply your function to your student id and state the output.**