EEE2020-03 **Spring 2021**

Assignment 1: C++ Class

Due: Monday, Mar. 29, 2021, 11:59PM

1 Introduction

- The objective of this assignment is to implement a C++ class based on the examples of lecture slides.
- The lecture slides of `1_cpp_programming.pdf` covered many basic concepts of C++, such as constructor and destructor, public and private members, namespace, pointer and reference, dynamic memory allocation, etc.
- You will practice the basics of C++ programming by implementing `class array_t`. The class you will have to implement in this assignment is slightly different from what is described in the lecture slides, so carefully read the instructions in this document.
- This document assumes that you have already installed Ubuntu. If you have not done so, refer to installation directions explained in `linux_guide.pdf`.
- In case you own an Apple Mac (of course, running Mac OS), you can directly work on the Mac terminal without installing Ubuntu. Ubuntu and Mac OS share many commonalities originated from the history of Linux and Unix, while Windows is the different lineage of OS.
- Either in Ubuntu or Mac, open the terminal and enter the following `wget` command to download a copy of skeleton code compressed into `class.tar`.

```
$ wget https://icsl.yonsei.ac.kr/wp-content/uploads/class.tar
```

- Then, decompress the downloaded `tar` file, and go to the `class/` directory as follows.

```
$ tar xf class.tar
$ cd class/
```

- In the `class/` directory, type `ls` to find the following list of files.

```
$ ls
array.cc  array.h  func.cc  func.h  main.cc  tar.sh  Makefile
```

- From the listed files, `array.cc` and `array.h` are the ones you will have to work on.
- You are allowed to change other files for testing and validation purposes, but note that they will revert to the original state when your assignment is graded. Do not create extra dependencies between `array.cc` or `array.h` and other files.
- To compile the code, you can simply type `make` in the terminal. `Makefile` that comes along with the skeleton code has automated all the compilation scripts for you.

```
$ make
g++ -Wall -Werror -g -o array.o -c array.cc
g++ -Wall -Werror -g -o func.o -c func.cc
g++ -Wall -Werror -g -o main.o -c main.cc
g++ -o array array.o func.o main.o
```

- Executing the program should print the following lines.

```
$ ./array
a1      a2
----------
```

- To clean up the directory and rebuild the code, you can type `make clean` and then `make` again. The `make clean` command does not change C++ files but only deletes the executable program.

```
$ make clean
rm -f array.o func.o main.o array

$ make
g++ -Wall -Werror -g -o array.o -c array.cc
g++ -Wall -Werror -g -o func.o -c func.cc
g++ -Wall -Werror -g -o main.o -c main.cc
g++ -o array array.o func.o main.o

$ ./array
a1      a2
----------
```

2 Implementation

- The following shows the contents of `array.h` and explains the implementation guideline.
- This file defines `class array_t`, but it misses a copy constructor and destructor of the class.
- Write the missing copy constructor and destructor definitions in the class. They are the only things you have to write in this file.
- Explanations of `reserve()` and `push_back()` functions will be made later with `array.cc`. Do not modify these lines of code in the file.
- The `[]` operator returns the reference of array element at the specified index. `size()` and `capacity()` functions return `num_elements` and `array_size`, respectively. These functions are already implemented, so you do not have to re-write them in `array.cc`.
- Member variables in the `private` part of class are complete. Do not modify them (i.e., `ptr`, `num_elements`, and `array_size`).

```
#ifndef __ARRAY_H__
#define __ARRAY_H__

typedef unsigned data_t;        // data_t represents unsigned int.

class array_t {
public:
    // Class constructor
    array_t();


    /* Assignment 1:
       Write a copy constructor and destructor of the class.
     */
```

```
    // Allocate a memory space for the specified number of elements.
    void reserve(const size_t m_array_size);
    // Add a new element at the end of array.
    void push_back(const data_t m_value);
    // Reference operator
    data_t& operator[](const size_t m_index) const { return ptr[m_index]; }
    // Return the number of elements in the array.
    size_t size() const { return num_elements; }
    // Return the size of allocated memory space.
    size_t capacity() const { return array_size; }

private:
    data_t *ptr;                // Pointer to an array
    size_t num_elements;        // Actual number of elements in the array
    size_t array_size;          // Allocated size of array
};

#endif
```

- The next shows the contents of `array.cc`. Likewise, this file misses the copy constructor and destructor. The `reserve()` and `push_back()` functions are incomplete as well.
- The constructor initializes `num_elements` and `array_size` as 0. It means that no array is created in the constructor, and thus `ptr` is set as a null pointer. The constructor is complete. Do not change it.
- Write the missing copy constructor and destructor functions. The copy constructor should copy all the contents of another class instance of the same type, and the destructor must deallocate the memory space pointed by `ptr`.
- The `reserve()` function allocates a memory space for the specified number of elements (i.e., `m_array_size`). In this assignment, assume that the `reserve()` function is called only when no array has been previously created; `num_elements` and `array_size` are zeros, and `ptr` is a null pointer when this function is called. The memory space must be allocated by `malloc()` not by `new`, since the `new` operator fills in the space by creating `data_t` instances. The `reserve()` function only allocates the memory space.
- The `push_back()` function inserts a new value (i.e., `m_value`) at the end of array, and `num_elements` and `array_size` must be accordingly updated. Assume that `reserve()` has allocated an enough memory space prior to the invocation of `push_back()` so that there are always empty slots to insert the new value. No error checking is necessary in this assignment.
- Assumptions made for `reserve()` and `push_back()` functions will significantly simplify the implementations of them. You do not have to worry about corner cases in this assignment (e.g., array is full when `push_back()` is called, or `reserve()` is called when an array has already been created).

```
#include <cstdlib>
#include <new>
#include "array.h"

// Class constructor
array_t::array_t() :
    ptr(0),
    num_elements(0),
    array_size(0) {
    // Nothing to do
}


/* Assignment 1:
   Write a copy constructor and destructor of the class,
   and complete reserve() and push_back() functions.
```

```
 */


// Allocate a memory space for the specified number of elements.
void array_t::reserve(const size_t m_array_size) {

}

// Add a new element at the end of array.
void array_t::push_back(const data_t m_value) {

}
```

- Finally, `main()` looks as follows and tests various features of class operations.
- It first creates a class instance named `a1`. Calling the `reserve()` function of `a1` allocates a memory space for `N = 30` elements.
- A series of 30 Fibonacci numbers (i.e., 1, 1, 2, 3, 5, 8, 13, $\cdots$) is pushed back into the array one at a time.
- Another class instance named `a2` is created as a copy of `a1`. The array is sequentially scanned, and non-prime numbers are set to zeroes.
- Lastly, the contents of `a1` and `a2` are printed out for validation.

```
#include <iostream>
#include "array.h"
#include "func.h"

#define N 30

using namespace std;

int main(void) {
    // Create a class instance, a1.
    array_t a1;

    // Allocate the memory space of a1 for N elements.
    a1.reserve(N);

    // Push back a sequence of Fibonacci numbers into a1.
    for(unsigned i = 0; i < N; i++) {
        a1.push_back(fibonacci(i));
    }

    // Create a class instance, a2, as a copy of a1.
    array_t a2 = a1;

    // Mask out non-prime numbers in a2.
    for(size_t i = 0; i < a2.size(); i++) {
        if(!is_prime(a2[i])) {
            a2[i] = 0;
        }
    }

    // Print the elements of a1 and a2.
    cout << "a1\ta2" << endl << "----------" << endl;
    for(size_t i = 0; i < a1.size(); i++) {
        cout << a1[i] << (a1[i] ? "" : "\b ") << "\t"
```

```
            << a2[i] << (a2[i] ? "" : "\b ") << endl;
    }

    return 0;
}
```

- Executing the code should give the following output.

```
$ ./array
a1      a2
----------
1
1
2       2
3       3
5       5
8
13      13
21
34
55
89      89
144
233     233
377
610
987
1597    1597
2584
4181
6765
10946
17711
28657   28657
46368
75025
121393
196418
317811
514229  514229
832040
```

- Even if your code compiles and runs to completion with correct results, you need to double-check the code via `valgrind` as follows to confirm no memory leaks.

```
$ valgrind ./array

...

==40205== All heap blocks were freed -- no leaks are possible

...
```

- In Mac OS, `valgrind` is not well supported. You may use `leaks` instead to check memory leaks as follows.

```
$ leaks --atExit -- ./array

...
```

```
Process 31661: 0 leaks for 0 total leaked bytes.

...
```

3 Submission

- When the assignment is done, execute the `tar.sh` script in the `class/` directory as follows.
- It will ask your student ID as input and then compress the `class/` directory into a `tar` file named after your student ID such as `2020140000.tar`.

```
$ ./tar.sh
Enter your 10-digit student ID: 2020140000

rm -f array.o func.o main.o array
a class
a class/main.cc
a class/array.cc
a class/func.cc
a class/Makefile
a class/array.h
a class/func.h
a class/tar.sh

$ ls
2020140000.tar  array.cc  func.cc  main.cc
Makefile        array.h   func.h   tar.sh
```

- Upload the tar file (e.g., `2020140000.tar`) on Y-EdNet. Do not rename the `tar` file or C++ files included in it.
- If you use Ubuntu on VirtualBox, there must be a pre-installed FireFox web browser for you to access Y-EdNet.
- If you use WSL in Windows 10, Ubuntu files are located at `C:\Users\[USERNAME]\AppData\Local\Packages\CanonicalGroupLimited.UbuntuWindows_[CODE]\LocalState\rootfs\[USERNAME]`. Actual path names may differ depending on Ubuntu versions and your usernames.

4 Grading Rules

- The following is the general guideline for grading. 30-point scale will be used for this assignment. The minimum score is zero, and negative scores will not be given. Grading rules are subject to change, and the grader may add a few extra rules for fair evaluation of students' efforts.

  -3 points: A submitted `tar` file is renamed and includes redundant tags such as `project1`, student name, etc.

  -5 points: A program code does not have sufficient amount of comments. Comments in the skeleton code do not count. You must make an effort to clearly explain what each part of your code intends to do.

  -10 points: The code compiles, runs to completion, and prints out correct results, but it has memory leaks.

  -15 points: The code compiles and runs with correct results, but it crashes at the end with errors.

  -15 points: The code compiles and runs to completion, but it prints a few incorrect results.

  -25 points: The code does not compile or fails to run (e.g., no or completely wrong results), but it shows substantial efforts to complete the assignment.

  -30 points: No or late submission. The following cases will also be regarded as no submissions.

  * Little to no efforts in the code (e.g., submitting nearly the same version of code as the skeleton code) will be regarded as no submission. Even if the code is incomplete, you must make substantial amount of efforts to earn partial credits.

* Fake codes will be regarded as cheating attempts and thus not graded. Examples of fake codes are i) hard-coding a program to print expected outputs to deceive the grader as if the program is correctly running, ii) copying and pasting random stuff found on the Internet to make the code look as if some efforts are made. More serious penalties may be considered if students abuse the grading rules.

Final grade = F: The submitted code is copied from someone else. All students involved in the incident will be penalized and given F for the final grades irrespective of assignments, attendance, etc.

- Your teaching assistant (TA) will grade your assignments. If you think your assignment score is incorrect for any reasons, feel free to discuss your concerns with the TA. In case no agreement is made between you and the TA, elevate the case to the instructor to review your assignment. Refer to the course website for the contact information of TA and instructor: `https://icsl.yonsei.ac.kr/eee2020`
- Begging partial credits for no valid reasons will be treated as a cheating attempt, and such a student will lose all scores of the assignment.