Programming Fundamentals Midterm ECE244, Fall 2022

Duration: **2 hours**, Examiner: D. Yuan, SK Rahman, S. Bhadra

Examination Aids: This is an open book exam.

If any of the questions appear unclear or ambiguous to you, then make any assumptions you need, state them and answer the question that way. If you believe there is an error, state what the error is, fix it, and respond as if fixed.

Be brief and specific. Clear, concise answers will be given higher marks than vague, wordy answers. Marks will be deducted for incorrect statements.

You will receive 20% of the marks of each (sub) question if you leave the answer blank.

*There are **18** total numbered pages, **10** Questions.*

Name: _____

Student Number: _____

	Total Marks	Marks Received
Question 1	8	
Question 2	5	
Question 3	5	
Question 4	5	
Question 5	12	
Question 6	10	
Question 7	10	
Question 8	7	
Question 9	8	
Question 10	30	
Total	100	

Q1: [8 marks, 2 marks each] Multiple choice (with some free mark questions), no explanation needed.

(1) How many bits are there in a Byte?
- (a) 8
- (b) 4
- (c) 2
- (d) 16
- (e) None of the above

(a)

(2) What programming language do we learn in ECE244?
- (a) Python
- (b) Memba
- (c) Ruby
- (d) Loonie
- (e) C++
- (f) Java
- (g) Rust
- (h) Dust

E

(3) On a 32-bit machine, the size of a pointer variable has:
- (a) 2 Bytes
- (b) 4 Bytes
- (c) 6 Bytes
- (d) 8 Bytes
- (e) 16 Bytes
- (f) None of the above

B

(4) Consider the following C++ program. There are three statements in the main function: A, B, and C. Which of these three statements will trigger compile-time errors?

```
class ECE244 {
private:
  int num_student;
  int num_TA;
  int get_num_instructor();
public:
  int num_instructor;
```

```
  int get_num_student();
  int get_num_TA();
};

int main() {
  ECE244 year2022;
  int num_student = year2022.num_student;              // Statement A
  int num_TA = year2022.get_num_TA();                  // Statement B
  int num_instructor = year2022.get_num_instructor(); // Statement C
  return 0;
}
```

A, C

Q2 (5 marks): When you compile the following program, what happens? If there is an error, explain what the error is (one sentence max).

```
#include <iostream>
using namespace std;
int main () {
  hello(1);
  return 0;
}

void hello(int i)
{
  cout << “Hello!” << i << endl;
  return;
}
```

Answer:
There will be a compilation error because hello() is called before it's **declared.**

Q3 (5 marks): Consider the following C++ function:

```
void AvadaKedavra(int n) {
  int size = n + 1;
  int* q = NULL;
  for (int i = 0; i < 3; ++i) {
    q = new int [size];
  }
}
```

If somewhere in your main function you call **`AvadaKedavra(1)`**. Based on the memory layout discussed during the lecture, answer this question: from the time this function starts to execute to the time right before it returns, how many bytes are newly allocated on the stack and the heap, respectively?

You may assume:
i. all variables are put in the main memory.
ii. an int takes 4 bytes
iii. we have a 32-bit machine

Marking scheme: if final answer is correct for stack but not (n, size, q), except for the situation when they only writes down the final answer as 12B: -1
Heap the same: if explanation wrong even with the right final answer: -1

Answer:
Stack:
n: 4B
size: 4B
q: 4B
i: 4B
Total: 16B

Heap:

6 integers: 24B

Marking: 3 marks for Stack, 2 marks for Heap.

Q4 (5 marks): Consider the following program. Write down the output.

```
#include <iostream>

using namespace std;

void increment(int& a) {
  a = a + 1;
}

void decrement(int a) {
  a = a - 1;
}

void doubling(int* a) {
  *a = (*a) * 2;
}

int main() {
  int a = 3;

  increment(a);
  cout << a << endl;

  decrement(a);
  cout << a << endl;
```

```
  doubling(&a);
  cout << a << endl;

  return 0;
}
```

Answer:
4
4
8

Marking: -2 for each mistake.

Q5 (12 marks, 2 marks each sub-question): Consider the following program.

```
#include<iostream>
#include<fstream>

using namespace std;

int main () {
  int a;
  ifstream inFile;
  inFile.open ("input.txt");
  if (inFile.fail()){
    return 1;
  }
  while (1) {
    inFile >> a;
    if (inFile.fail()) {
      cout << "failed.." << endl;
      inFile.clear();
      inFile.ignore(100, '\n');
      continue;
    }
    cout << "a = " << a << endl;
    break;
  }
  return 0;
}
```

Given the following contents of "input.txt", write the output.

(1) input.txt has:

```
1
2
3
```

Answer:
a = 1

(2) input.txt has

```
a32
b86
3
```

Answer:
failed..
failed..
a = 3

(3) input.txt has

```
a32
b86   3
```

Answer:
failed..
failed..
failed..
And it keeps repeating

Now you remove the infile.clear() from the code, so the program becomes:

```
#include<iostream>
#include<fstream>

using namespace std;

int main () {
```

```
  int a;
  ifstream inFile;
  inFile.open ("input.txt");
  if (inFile.fail()){
    return 1;
  }
  while (1) {
    inFile >> a;
    if (inFile.fail()) {
      cout << "failed.." << endl;
//      inFile.clear(); COMMENTED OUT
      inFile.ignore(100, '\n');
      continue;
    }
    cout << "a = " << a << endl;
    break;
  }
  return 0;
}
```

Answer each of the first 3 sub-questions again:

(4) input.txt has:

```
1
2
3
```

Answer:
a = 1

(5) input.txt has

```
a32
b86
3
```

Answer:
failed..
failed..

failed..
And it keeps repeating

(6) input.txt has

```
a32
b86    3
```

Answer:
failed..
failed..
failed..
And it keeps repeating

Marking: binary.

Q6 (10 marks): Pointer vs. Reference
Compared to C, passing by reference is introduced in C++. Both of the following two functions can be used to swap the value of two integers:

```
void swap_by_p(int* a, int* b); // swap version.1
void swap_by_r(int& a, int& b); // swap version.2
```

a. **[6 marks]** Write the implementations for these two functions (no more than 4 lines of code for each function)

```
void swap_by_p(int* a, int* b) {




}

void swap_by_r(int& a, int& b) {




}
```

Answer:

```
void swap_by_p(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void swap_by_r(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Marking: 3 marks for each function; 1 mark for each stmt.

b. [2 marks] If given two int variables x and y, write a function call that swaps the value of x and y, using swap_by_p.

Answer: swap_by_p (&x, &y);

c. [2 marks] If given two int variables x and y, write a function call that swaps the value of x and y, using swap_by_r.

Answer: swap_by_r (x, y);

Q7 (10 marks): Incremental compilation

Ellie writes a program to make a simple database for ECE students who like drinking soy milk from 2T2 to 2T6. She designs two classes: student and ECE, and puts them into different files below. The main function is in the main.cpp.

ECE.h

```
#ifndef ECE
#define ECE

#include "student.h"

class ECE {
        …
};

#endif
```

main.cpp

```
#include "ECE.h"
#include "student.h"

int main()  {
        …
};
```

student.h

```
#include <string>

class Student {
        …
};
```

ECE.cpp

```
#include "ECE.h"

ECE::ECE()  {
        …
};
```

student.cpp

```
#include "student.h"

Student::Student ()
{
        …
};
```

a. [2 marks] Ellie tries to compile this program with g++. What's the Unix (i.e., terminal) command that compiles the entire program using one command, which generates an executable called `small_database.exe`?

```
g++ ECE.cpp student.cpp main.cpp -o small_database.exe
```

b. [2 marks] However, it fails to compile. Can you point out the compile-time error and fix this error for her?

No header guard in student.h

c. [4 marks] With your help, Ellie has fixed the compile-time error. Now, Ellie wants to use separate compilation learned from ECE244 to compile her project. Write down all the Unix commands necessary to separately compile the above files and generate an executable small_database.exe.

```
g++ -c student.cpp -o student.o

g++ -c ECE.cpp -o ECE.o

g++ -c main.cpp -o main.o

g++ student.o ECE.o main.o -o small_database.exe
```

d. [2 marks] Ellie then changes some code in ECE.cpp. Write down the Unix commands necessary to regenerate the executable by compiling the smallest number of files needed. Assume the generated executable is called small_database.exe.

```
g++ -c ECE.cpp -o ECE.o

g++ student.o ECE.o main.o -o small_database.exe
```

Q8 (7 marks) Operator Overloading

Vector is widely used in the engineering and science world. Suppose we create a class called vector_2d. It can be used to represent a 2D vector, with x and y as its values. The partial code of this vector is shown below.

```
class vector_2d {
```

```
private:
  double x;
  double y;

public:
  vector_2d() {
    x = 0;
    y = 0;
  }

  vector_2d(double x_, double y_) {
    x = x_;
    y = y_;
  }

  double get_x() const {
    return x;
  }

  double get_y() const {
    return y;
  }

  // Add overloaded operator+= here
}
```

Implement `operator+=` for `vector_2d`, as a member function. If `a = <x1, y1>` and `b = <x2, y2>`, after `a += b`, `a` becomes `<x1 + x2, y1 + y2>` whereas `b` is unchanged. Write no more than 5 lines of code.

Answer:

```
vector_2d& vector_2d::operator+=(const vector_2d& rhs) {
    x += rhs.x;  // 1 mark
    y += rhs.y;  // 1 mark
    return *this;  // 1 mark
```

```
}
```

NOTE: the return type cannot be void, because we need to support chained +=.
Example: a += b += 2;
which is equivalent to:
b += 2;
a += b;

Marking:

- Correct return type (vector_2d&): 2 marks
- const: 1 mark
- Pass by reference: 1 mark
- 1 mark for each statements in the function body

Q9 (8 marks).Write down the standard output of the following program. Remember to write two "Check Point", since partial marks are given based on these "stop points". You might find it helpful to write down the memory layout.

```
#include <iostream>
using namespace std;

int i[5] = {0, 2, 4, 6, 8};
int* p;

void foo() {
  cout << *p << endl;
  ++(*p);
  ++p;
}

void bar() {
  for (int i = 0; i < 3; ++i) {
    foo();
  }
}

int main() {
  p = i;
  bar();
  cout << "Check Point 1" << endl;
```

```
  p = i;
  foo();
  cout << "Check Point 2" << endl;

  return 0;
}
```

0

2

4

Check point 1

1

Check point 2

Marking:

-2 for each mistake (including value and/or order)

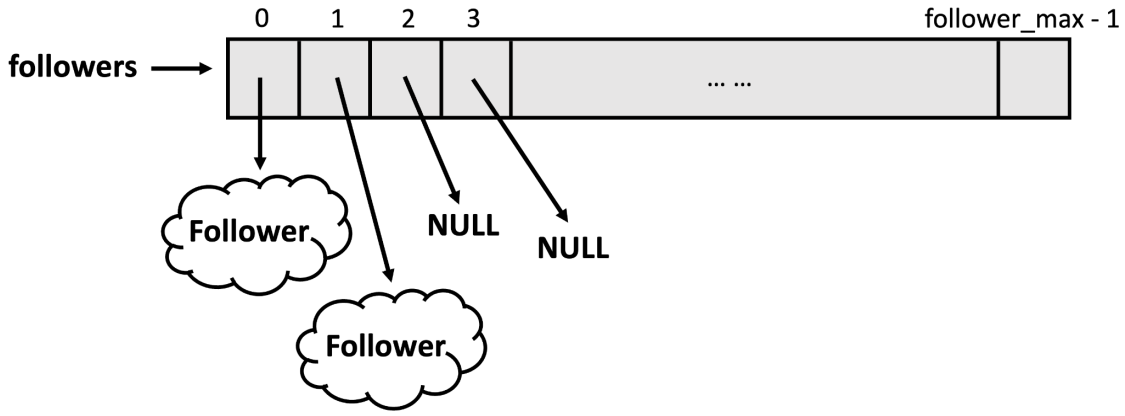
Q10 (30 marks). Programming

A Vtuber is an online entertainer who posts videos on Vtube. A Vtuber will have followers on Vtube. As a programmer from Vtube, you are asked to implement a class for Vtuber. The class definition and description are described below.

```
#include <string>

using namespace std;

class Follower {
private:
    string name;
    int age;
public:
    Follower(const string& _name, int _age) {
        name = name_;
        age = age_;
```

```
    }
    string get_name() const {
        return name;
    }
    int get_age() const {
        return age;
    }
};

class Vtuber {
private:
    // Vtuber Name
    string name;

    // Follower array with a variable size, each element should be a dynamically
    // allocated object of class Follower.
    Follower** followers;

    // The size of follower array.
    int follower_max;

    // Number of followers
    int follower_num;

public:
    Vtuber(const string& _name);
    ~Vtuber();
    void insert_follower(const string& follower_name, int follower_age);
    void remove_follower(const string& follower_name);
};
```

Specifically, Vtuber's **`followers`** member variable is an array of pointers, each pointer pointing to a **`Follower`** object. The following graph illustrates it.



a. **[6 marks]** Implement the constructor for `Vtuber`. `Vtuber` **name** should be initialized by `_name`, and **`follower_max`** should be initialized to 2. In addition, you should allocate an array called **`followers`** using **`new`**, with an initial size of 2 (the value of **follower_max)**. Every element in this array should be a pointer to an object of class **Follower** and initialize all these pointers to NULL.

```
Vtuber::Vtuber (const string& _name) {

}
```

Answer:

```
Vtuber::Vtuber (const string& _name) {
    name = _name;
    follower_max = 2;
    follower_num = 0;
    followers = new Follower*[follower_max];
    for (int i = 0; i < follower_max; i++) {
        followers[i] = NULL;
    }
```

```
}
```

Marking:

- 2 marks for the correct allocation of followers
- 3 marks for initializing followers to NULL
- 1 mark for others

b. Every Vtuber in Vtube can get new followers or lose their current followers. This is implemented by two methods: **`insert_follower`** and **`remove_follower`**. Now you are asked to implement these two methods:
   i. **[10 marks]** For `insert_follower`, a follower name and follower age are given. You need to create a dynamically allocated object of `Follower` and insert it into the **`followers`** array (in the first available location), using **new** operator. If the array is full, you should double **follower_max**, allocate a new follower array, and move all the data into this new array, and insert the new follower. Write the code below.

```
void Vtuber::insert_follower(const string& follower_name,
       int follower_age) {
```

```
}
```

Answer:

```
void Vtuber::insert_follower(const string& follower_name,
       int follower_age) {
    follower_num++;
    for (int i = 0; i < follower_max; i++) {
        if (followers[i] == NULL) {
            followers[i] = new Follower (follower_name, follower_age);
            return;
        }
    }

    Follower** new_followers = new Follower*[2*follower_max];
    for (int i = 0; i < follower_max; i++) {
        new_followers[i] = followers[i];
        new_followers[i + follower_max] = NULL;
    }

    new_followers[follower_max] =
          new Follower (follower_name, follower_age);
    delete [] followers;

    followers = new_followers;
    follower_max *= 2;
    return;
}
```

Marking:

- 3 marks on correct insertion without array expansion
  - 1 mark for correct allocation
  - 1 mark on correct looping
  - 1 mark on if condition checking
- 7 marks on correct insertion with array expansion (multiple ways to do it)
  - 2 marks on initializing newly allocated members to NULL
  - 2 mark on copying

- 1 mark on allocation
- 2 marks on correct deleting followers (-1 if not using [])

- Any other mistakes: -1 each.

ii. **[8 marks]** For `remove_follower`, a follower name is given. If there is any follower in the array matching the name, you should remove it and free its memory using **delete**. You can assume the follower names are all unique.

Note: Your code should not have any memory leaks!

```
void Vtuber::remove_follower(const string& follower_name) {

}
```

Answer:

```
void Vtuber::remove_follower(const string& follower_name) {
    for (int i = 0; i < follower_max; i++) {
        if (followers[i] == NULL) // 3 marks on skipping NULL members
            continue;

        if (followers[i]->get_name() == follower_name) { // 1 mark
            follower_num--; // 1 mark
```

```
            delete followers[i]; // 2 mark
            followers[i] = NULL; // 1 mark
            break;
        }
    }
    return;
}
```

Marking:

- see comment

c. **[6 marks]** Implement the destructor for the `Vtuber` class. You should free all the dynamically allocated objects using **delete**. Remember to be consistent with your previous implementation, as the entire program should not trigger any segmentation fault.

```
Vtuber::~Vtuber() {




}
```

Answer:

```
Vtuber::~Vtuber() {
    for (int i = 0; i < follower_max; i++) {
        delete followers[i]; // delete NULL is safe;
    }
    delete [] followers;
}
```

Marking:

- **3 marks for delete [] followers**
  - **-2 if not using []**
- **3 marks for deleting each follower[i]**