**University of California, Los Angeles December 08th, 2021 PIC 10A: Introduction to Programming Fall 2021, Section 3 Instructor: Minh Pham**

# Final

Read all of the following information before starting the exam:

1. The test is open book, open note, open Visual Studio 2019, **not open internet**. But you are allowed to use CCLE to ask general questions.
2. Only use material that I taught you in this class. Do not use material beyond this class. No collaboration is allowed.
3. You have two options for submission:
   a. Download this file, and write your solutions in the space below the questions, convert this file to pdf or image files, or take pictures of these pages.
   b. Type your solutions in MS word, then convert it to a pdf file.
4. When submitting through Gradescope, please match your solution page with the outline. 5. Do not spend too much time on any problem. Read them all through first and attack them in the order that allows you to make the most progress.
6. This test has **10 questions** which are worth **120 points**.
7. Please follow instructions closely and attempt all problems. Incomplete answers still get partial credit while no attempt definitely gets zero.

## Statement of Academic Honesty:

For this exam, I make the following truthful statements:

● I have not received, I have not given, nor will I give or receive, any assistance to another student taking this exam, including discussing the exam with students in another section of the course. ● I will not use any non-instructor approved electronic device to assist me on an exam. ● I will not plagiarize someone else's work and turn it in as my own.

By signing below, I declare that this exam represents my own work in accordance with University policy.

**Name:__Le'on_Norfleet______________ Student ID:__305771450____**
**Signature:** [illegible] **Discussion session:__3D___**

**Question 1 (4pts)**. These three expressions `1.0/4.0`, `1.0/4`, `1/4.0` all result in a floating-point value `0.25`. However, the expression `1.0/4.0` is a better practice than the others. Explain (you only need **1-2 sentences to explain**, do not write too much).

**1.0/4/0 is a better practice than the others because both numbers divided are floating-point. The other examples are dividing a floating-point and an integer which may lead to confusion.**

**Question 2 (4pts):** Write one line of code that generates a floating-point value (of type double) between `-5.0` and `3.0`. Use `rand()` and `RAND_MAX` only. Assume libraries were included.

```
double x = ______?______;
```

```
double x = rand() * 8.0 / RAND_MAX - 5.0;
```

�� + 1

**Question 3 (8pts):** We know that if we divide the integer 7 by 7, the remainder is 1. A student wants to prove it computationally. He writes a C++ program as follow

```
int n;
cout << "Please enter a positive integer n (less than 25): ";
cin >> n;
long long result = pow(7,n) + 1;
cout << "(7^n + 1)%7 = " << result%7 << endl;
```

The program should print out 1 every time; however it does not give expected results. Explain all the problems (run-times errors) of this code. You can assume that the cmath library is provided and that the user always enters valid inputs (positive integers less than 25).

**The problem with this code is that it is using extremely large values. The computer won't hold the exact, perfect number. So the larger numbers become, the more inaccurate they are, which eventually causes errors at powers such as 21. Also, at power 23, result goes over the numerical limit of long long.**

**Question 4 (6pts):** True or false. No need explanation

a. A function can never return an array.

   Answer: **True**

b. However, a function can return an array via a pointer, i.e. return a pointer to an array. This array does not need to be a dynamic array.

   Answer: **True**

c. An alternative way is creating a function that returns an object whose data member is an array.

   Answer: **False**

d. One difference between arrays and vectors is that arrays are always passed by reference while vectors are always passed by values.

   Answer: **False**

**Question 5 (4pts):** Is there any compile-time error or run-time error in the following code? If there is one, show which line causes the error, and explain.

```
void fun(double a[], double value){
    *(a+1) = value*value*value;
}

int main(){
    double x = 3.0;
    double* x_ptr = &x;
    fun(x_ptr, 5);
    cout << "*x_ptr = " << *x_ptr << endl;
    cout << "*(x_ptr+1) = " << *(x_ptr+1) << endl;
}
```

**Neither x_ptr nor x are arrays, so calling the fun function and using x_ptr as the array parameter is the run-time error. This is line 4 of the int main function where fun is being called.**
**The compile-time error is setting the address of *(&x + 1) equal to 125, which is a random part of the memory(also on line 4 of int main where fun is being called).**

**Question 6 (10pts):** Suppose the array primes, defined as

```
double primes[] = {2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37};
```

starts at memory location 30000. Answer the following:

1. (6pts) What are the values of (i.e what is displayed when you `cout` these terms)
   a. `primes + 4   = 30000 + 4 * 8 = 30032`
   b. `&primes[3]   = 30000 + 3 * 8 = 30024`
   c. `*(primes + 5)= 13`
   d. `*primes + 5 = 2 + 5 = 7`
   e. `*(&primes[4] + 3) = 23`
   f. `&*(primes + 2) = 30000 + 2 * 8 = 30016`

2. (4pts) Explain why `&*primes[2]` causes compile-time errors in **1-2 sentences** only.

   **&*primes[2] causes compile-time errors because the address of 5 cannot be accessed; you cannot reference a dereferenced pointer at the same time**

**Question 7 (14pts):** This problem refers to hw8_1. Use the class ComplexNumber to do the following

1. (4pts) Create these ComplexNumber objects. Make them `const` objects
   a. `z1 = 3.0 + 4.0i;` `const ComplexNumber(3.0, 4.0)`
   b. `z2 = 2.0 - 5.2i;` `const ComplexNumber(2.0, -5.2)`
   c. `z3 = -1.5 + 6.0i;` `const ComplexNumber(-1.5, 6.0)`
   d. `z4 = 2.0 + 3.0i;` `const ComplexNumber(2.0, 3.0)`

2. (10pts) Compute the following ComplexNumber objects. Note that you can't modify `z1, z2, z3, z4` because they are `const` objects.
   a. $z5 = (z2 + \overline{z3}) * z4 - z1$, where $\overline{z3}$ is the conjugate of $z3$. Do not create extra ComplexNumber objects (except the required one),

   z5 = -0.5 + 11.2i

   b. $z6 = 3 * z2 + |z1|$ where $|z1|$ is the norm of $z1$. Recall a real number is a complex number where the imaginary part is 0.
   z6 = 11 - 15.6i

**Question 8 (20pts):** Write a function

```
vector<double> remove_negative (vector<double> & vec)
```

that moves all negative elements of `vector<double> vec` to a new `vector<double>` and returns this new vector. Your function should not change the order of positive elements and negative elements (i.e. you are removing an element from an ordered `vector`). For example:

```
vector<double> vec = {1,-2,3,4,-6,5,-5,8,7,8};
vector<double> negative_part = remove_negative(vec);
```

After the function call, `negative_part={-2,-6,-5}` and `vec={1,3,4,5,8,7,8}` respectively.

Please only use material & member functions of vectors that I taught in this class, such as `pop_back(),push_back(), size().`

```
vector <double> remove_negative(vector<double>& vec) {

    vector <double> copy;

    for (int i = 0; i < vec.size(); i++) {

        if (vec[i] < 0) {

            copy.push_back(vec[i]);

            for (int j = i; j < vec.size() - 1; j++) {

                vec[j] = vec[j + 1];

            }

            vec.pop_back();

        }

    }

    return copy;

}
```

**Question 9 (20pts)**. Write a function:

`char* merge (char str1[], char str2[])`

that merges two C-strings `str1` and `str2`, alternating characters from both C-strings (`str1` first). If one C-string is shorter than the other, then alternate as long as you can and then append the remaining characters from the longer C-string. For example, if `str1 = "PC0-"` and `str2 = "I1AExam"` then `merge` returns a pointer to a dynamic C-string `"PIC10A-Exam"`

Requirement: Do not use any Cstring functions (`strcpy`, `strcat`,...) except `strlen`. Do not use pointer arithmetic in this problem. Use the **square bracket []** instead. You need to create a dynamic `char` array that has the **exact size** to merge `str1` and `str2`.

```
char* merge(char str1[], char str2[]) {

    int n = strlen(str1) + strlen(str2);

    int len1 = strlen(str1);

    int len2 = strlen(str2);

    char* temp = new char[n];

    char* final = new char[n];

    int even = 0;

    int odd = 1;

if(len1 > len2) {

    for (int i = 0; i < len2; i++) {

        temp[even] = str1[i];

        temp[odd] = str2[i];

        even += 2;

        odd += 2;

    }

}
```

```
    else {
    for (int i = 0; i < len1; i++) {
           temp[even] = str1[i];
           temp[odd] = str2[i];
           even += 2;
           odd += 2;
    }
}
     int count = strlen(temp);
     if (len1 > len2) {
        for (int j = 0; j < len1 - len2; j++) {
              temp[count + j] = str1[len2 + j];
        }
     }
     else {
         for (int j = 0; j < len2 - len1; j++) {
               temp[count + j] = str2[len1 + j];
     }
}
     return temp;
}
```

**Question 10 (30pts)**. This problem refers to the example `BankAccount` in our lectures.

1. (16pts) Create a **dynamic array** of type BankAccount and size 100. The i_th element of the array is set as follows
   - The instance variable `name` is set as `"account i"`. For example, when `i=40`, then the data member `name` is set as `"account 40"`. Note that `i` is in the range [1,100].
   - The instance variable `balance` is a random number in the range [1000,4000], and is a multiple of 100, i.e. `balance` only takes values 1000, 1100, 1200, …, 4000.

   **Use pointer arithmetic to access elements of the array. Do not use square brackets []**

```
BankAccount* BankArray = new BankAccount[100];

    srand(time(0) * 1.0);

    for (int i = 0; i < 100; i++) {

        int value = (rand() % 31 + 10)*100;

        string name = "account " + to_string(i + 1);

        (BankArray + i)->setName(name);

        (BankArray + i)->setBalance(value);

        (BankArray + i)->display();

    }
```

2. (14pts) Implement a (non-member) function `merge2Accounts(acc1,acc2,new_name)` that returns a new `BankAccount` object whose data member `name` is set to `new_name`, and data member `balance` is set to be the sum of these two `BankAccount` objects' `balances`. Then set `names` of `acc1` and `acc2` to `"inactive account"` and set their `balances` to 0.

   For example,

```
  BankAccount acc1("Adam 1", 1000);
  BankAccount acc2("Adam 2", 2000);
  BankAccount acc3 = merge2Accounts(acc1, acc2, "Adam 12"); // after
this call, acc1.name="inactive account", acc1.balance = 0 //
```

```
acc2.name="inactive account", acc2.balance = 0 // acc3.name="Adam
12", acc3.balance = 3000
```

You need to figure out the function prototype yourself.

```
void Merge2Accounts(BankAccount& acc1, BankAccount& acc2, string
new_name) {

   acc1.display();

   acc2.display();

   BankAccount acc3;

   acc3.setName(new_name);

   double new_balance = acc1.getBalance() + acc2.getBalance();

   acc3.setBalance(new_balance);

   cout << "Transfer Completed! Accounts Cleared." << endl;

   acc1.setName("inactive account");

   acc1.setBalance(0);

   acc2.setName("inactive account");

   acc2.setBalance(0);

   acc3.display();

   acc2.display();

   acc1.display();

}
```