Review for Midterm of CS 135, Fall 2023

- The exam will be in paper. You must take the midterm in class room unless you got approval to take the exam in a different classroom in campus.
  Note that if you need accommodation, please take the exam at accessibility office. Our classroom will be reserved for other classes after 10 AM. As a result, I cannot give you extra time or quiet environment.
- Closed book, closed notes. No Internet or computer or cell phone or other electronic devices.
- A review sheet https://maryash.github.io/135/slides/CheatSheet.pdf will be provided (courtesy Professor Maryash). You cannot use your own review sheets.
- No cheating. Work independently.
- During the exam, put electronic devices, books, and notes in your bag and zip it.
- Cannot go to bathroom when taking the exam.

1. What are the differences between variables and const variables?
   (1) Any variable is declared first before being used. Examples of usage include print out or put in right hand side expression to initialize or update a variable.
   (2) The difference between a variable and a constant variable is, the value of a variable can be updated from time to time. It is a like a cup that can hold different amount of water (values) at different time.
   By contrast, a constant variable can only be initialized once. It is like a dumb bell, once it is created, its value is fixed and cannot be changed.

2. Rules for naming variables (including function or method names) in C++.
   (1) Start with letters or underscore symbol _
   Normally in variables in user application do not start names with _. Only variables in system coding will start with _.
   (2) Followed by letters, _, or digits.
   (3) Do not use keywords in C++ like if, int, while, and so on.
   (4) By convention, a constant variable is named with all capital letters. For example, to declare a variable representing number of hours per day, since this value is fixed, we use the following:

```
const int HOURS_PER_DAY = 24;
```

3. Arithmetic operators are +, -, *, /, and % (remainder or modular). They follow the same rules of combination and precedence in mathematics.

Pay special attention to division operator /.

(1) When both numerator and denominator are integers, the quotient is also an integer. For example, expression 3 / 5 returns 0. This is similar to divide 3 pens among 5 students, and the number of pens each student gets is zero.

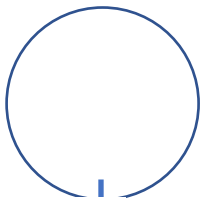
(2) When at least one of numerator or denominator contains decimals, the quotient can keep its decimal parts. For example,
3.0 / 5 returns 0.6.
1.0 * 3 / 5 returns 3.0 / 5 = 0.6.
(double) 3 / 5 forces 3 to change to double type. Here (double) means type cast, which forces the immediately followed variable or value – in this example, number 3 -- to become a double type, which is used to store values with decimal parts.

(double) (3 / 5) forces 3 / 5 into a double type. Since 3 / 5 returns 0 (integer division), so (double) (3 / 5) still returns 0.

1.0 * (3 / 5) returns 1.0 * 0 = 0.

Pay attention to % (remainder or modular operator). For example, result of 3 % 5 is 3, which can be interpreted as dividing 3 pens among 5 students, none of the students get any pen, and the number of pens left is 3.

Remainder operator is useful in "circle back" operation. For example, suppose now is 6 AM, what time was 8 hours AGO? Since number of hours we represent is 12, then (6 + 12 – 8) % 12 = 10. Similarly, what is 20 hours AFTER 6 AM? Then (20 + 6) % 12 = 2.



What is 20 hours BEFORE 6 AM, then it is (6 – 20 + 24) % 12 = 10. Where 24 is a multiple of 12.

This % operator is useful in Lab 6 string encrypt and decrypt.

4. Assignment operator = runs from right to left. Evaluate the right hand expression first, then use its value to update the left hand variable.

For example,
int a = 5;
a += 2; //the same as a = a + 2;

After the above segment, a is changed to 7.

5. Do not confuse assignment operator = with equality operator ==. In assignment operator, the right-hand side expression is evaluated, then use that value to set left-hand side variable, afterwards, the whole expression has that value.
   Equality operator compares the left-hand side expression with the right-hand side expression. If both sides equal, then return true (1 in C++), otherwise, returns false (0 in C++).

```
int a = 5;
cout << (a += 2) << endl; //a += 2 returns 7,
                          //print 7

cout << (a == 7) << endl; //print 1
cout << (a == 5) << endl; //print 0
```

6. To generate a random int in [start, end], where start and end are both integers and start <= end, we can use the following statement.
   srand(time(NULL));
   //Set seed to generate a pseudo-random sequence, use only once.
   int value = rand() % (end – start +1) + start;
   //Generate a random int in [start, end] and put in variable value.
   //There are a total of (end – start + 1) integers in range [start, end].

   To generate a random floating number in [start, end], where start <= end.
   (1) Use 1.0 * rand() / RAND_MAX get a random floating point number in [0, 1].
   Question: can we use rand() / RAND_MAX?
   (2) Amplify the result by 1.0 * rand() / RAND_MAX * (end – start) to get a random floating point number in [0, end-start].
   (3) Shift the above result by start to get a random floating point number in [start, end].
   1.0 * rand() / RAND_MAX * (end – start) + start

7. Branch statement includes if-, if-else and nested if-else statement. Nested if-else statements are mutually exclusive. For example, given an int, it can be positive, negative, or zero. Another example is traffic light and letter grades. If conditions are not mutually exclusive, for example, citizenship (some people can have dual citizenship), use separated if- statement.

8. For repetition statement, pay attention to initialization, condition to stop, what to repeat in each loop, how to update loop variable.

   Some examples are in Labs 2, 3, 4, and class exercises on repetition statements.

   Syntax of while-statement

```
Initialize loop variable
while (condition)
{      //if loop body has one statement, can omit curly brackets.
       things-to-do-in-one-round (including update loop variable)
}
```

   Syntax of for-statement

```
for (initialize loop variable; condition; update loop variable)
{    //if loop body has one statement, can omit curly brackets.
     things-to-do-in-one-round
}
```

   Syntax of do-while statement

```
do
{
      things-to-do-in-one-round
} while (condition); //do not forget ; to end do-while statement.
```

   (a) while-statement and for-statement run entry-check. That is, check the condition is satisfied or not before entering the loop.
   (b) for-statement put initialization of loop variable and update loop variable explicitly into for-header. In for-statement, loop variable is updated after

things-to-do-in-one-round. For-statement is often used in situation when the number of repetitions is known in advance.

(c) do-while statement checks the condition at exit point. Statements in loop body of do-while statement runs at least once.

(d) do-while statement can be rewritten as a while-statement.

```
thing-to-do-in-one-round //run loop body once
while (condition)
 {
      thing-to-do-in-one-round
 }
```

do-while statement is used in situation when loop-body statement runs at least once. For example, find out number of digits of an integer by throwing away a digit a time and each integer has at least one digit.

The following code define a method to find out numbers of digits of an int.

```
#include <iostream>
#include <cassert> //assert
using namespace std;

int getNumDigits(int n); //return number of digits of n
int main()
{
   assert(getNumDigits(5) == 1);
   assert(getNumDigits(-5) == 1);
   assert(getNumDigits(0) == 1);
   assert(getNumDigits(69) == 2);
   assert(getNumDigits(-20) == 2);
   return 0;
}

int getNumDigits(int n)
{
   int numDigits = 0;
   do
   {
     n /= 10; //throw the least
```

```
        //significant digit from n
      numDigits++;
    } while (n != 0);

    return numDigits;
}
```

Another example for do-while statement is to enter an integer in [0, 100]. Since we need to take input at least once, we can use do-while statement.

9. When defining a function, remember to return in every possible branch and consider corner cases. For example, in the following isPrime function, need to consider when the input parameter is 1, 2, and 3 since for loop `for (int i = 2; i <= sqrt(num); i++)` only works for num >= 4.
   Reason: to enter the loop, we must have 2 <= sqrt(num). Then square both sides of the inequation, get num >= 4. As another example, to define a function to convert a numerical grade to a letter grade, consider the case when the input parameter is out of the range of [0, 100].

   - Return statement is very strong, it can be either in the form of return; if return type is void or return a_value_of_non_void_return_type;
   - Return statement leaves the current function and return to the caller function.
   - Compared with break statement, which can only be in the form of break;
   - Break statement jumps out the current repetition block or switch block where break; statement resides.

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

//if num is prime, returns true,
//otherwise, returns false.
bool isPrime(int num);

int main()
{
    cout << isPrime(5);
    return 0;
}

//A positive int is a prime if and only if
//it can be divided by 1 AND itself only.
```

```
//By definition, is 1 a prime int? No.
bool isPrime(int num)
{


    //for num, its trivial factors are 1 and num,
    //its POSSIBLE (candidates) of non-trivial factors are
    //2, ..., num/2.
    //We can shrink the range better.
    //Observation: non-trivial factors are paired.
    //For example, suppose num is 16,
    //then the non-trivial factors are 2, 4, 8
    //PAIRED non-trivial factors
    //2 * 8
    //4 * 4
    //8 * 2
    //The BIGGEST of smaller of PAIRED non-trivial factors is
    //sqrt(num).
    //Why? Suppose we have a pair of NON-trivial factors,
    //a and b, without loss of generality, we assume that a <=
b.
    //a * b is num,
    //a <= b
    //so a is <= sqrt(num);

    //special processing for num equals to 1,
    //when num is 1, we do not enter the for loop,
    //without the following if-statement,
    //when num is 1,
    //we do not go through for-loop, then we return true.
    //That means 1 is prime, contradict to definition.
    if (num <= 1)//negative integers are not prime, 1 is not
prime.
       return false;

    //for (int i = 2; i <= num/2; i++)
    //Why do we use sqrt(num) instead of num/2?
    //Suppose that num is 101, a prime num,
    //in the approach of
    //for (int i = 2; i <= num/2; i++)
    //    ...
    //we need to test every one of 2, 3, ..., 50.
    //But if we use
    //for (int i = 2; i <= sqrt(num); i++)
    //    ...
    //then we only need to test 2, 3, ..., 10 to find out that
    //101 is a prime.
```

```
    for (int i = 2; i <= sqrt(num); i++)
        if (num % i == 0) //i is an ACTUAL factor of num
           return false;

    return true; //return true ONLY after testing every non-
trivial
    //factors of num and none of them becomes an ACTUAL factor
of num.
    //like to test a batch of eggs.
    //for (every egg)
    //    if this current egg is bad
    //       return "bad quality"; //one bad means the whole
batch is bad
    //
    //return "good quality" //good quality for every egg
}
```

Also, do not use cout to print results to screen unless the function concentrates on printing. Reason: the screen is a unique resource, if every function prints to the screen without coordination from main function, the screen will be a mess.

Here is a running to test whether 25 is a prime or not.
Since 25 is not 1, we move to the next statement.

```
for (int i = 2; i <= sqrt(num); i++)
       if (num % i == 0) //i is an ACTUAL factor of num
           return false;

 return true;
```

Since sqrt(25) is 5, to run the above code when num is 25, replace sqrt(num) by 5.

```
for (int i = 2; i <= sqrt(num)5; i++)
      if (num % i == 0) //i is an ACTUAL factor of num
          return false;

 return true;
```

i = 2 (initialization)

Loop variable i	i <= 5?	For-body if (num % i == 0)	i++

		return false;	
2	yes	25 % 2 is not 0, do nothing	3
3	yes	25 % 3 is not 0, do nothing	4
4	yes	25 % 4 is not 0, do nothing	5
5	yes	25 % 5 is 0, return false	6

Similarly, to test whether 29 is a prime or not, we go through the following segment. Note that sqrt(29) is 5.xxx, a decimal number between 5 and 6. So the maximum integer i satisfying i <= sqrt(29) is 5.

```
for (int i = 2; i <= sqrt(29); i++)
     if (num % i == 0) //i is an ACTUAL factor of num
          return false;

 return true;
```

i = 2 (initialization)

Loop variable i	i <= sqrt(29)?	For-body if (num % i == 0) return false;	i++
2	yes	29 % 2 is not 0, do nothing	3
3	yes	29 % 3 is not 0, do nothing	4
4	yes	29 % 4 is not 0, do nothing	5
5	yes	29 % 5 is not 0, do nothing	6
6	no		

After moving out the loop, we run through its next statement

```
return true;
```

This means 29 is a prime.

10. For function, pay attention to pass by value and pass by reference. Pass by value is like to pass a duplicated copy and pass by reference is like to pass the original copy. Both of pass by value and pass by reference are of the same type, the only difference is add & after the type.
    For example, to apply for a visa, you need to send your original copy of passport. Use bool applyVisa(Passport& p); //suppose Passport is a type.
    To apply for a scholarship, you can send a duplicated copy of your grades. Use bool applyScholarship(Grade copy); //suppose Grade is a type
    For more examples, see Lab 5 Functions and Prime Numbers.

11. An array is a group of same-type elements residing in consecutive spaces. The name of an array also implies the address of its first element. The index of an array always starts from 0. The last index of an array is one less than its size.

12. Run operations for objects of a class. For example, given a string, how to find out the number of letters. For more exercises, see Lab 6.

13. File I/O. Also pay attention to file redirection in command line. For exercises, see Lab 3, Project 1, and E 8.1.

14. A pointer saves the address of the corresponding type. Address of a variable can be found by & operator. For example,

```
int a = 5;
int *p; //p save the address of an int
p = &a; //p points to a, or p saves the address of a.
        //*p can be used as an int, *p is an alias of a.
*p = 6; //a is 6 now
```

15. Pass by value vs pass by reference when defining and calling a function. For more details, see "What is the difference between int* example and int& example when we use function?" in questions answered for 10/5/23.

    As a sidenote, when an array is an input parameter for a function, since the name of an array also implies its initial address, or address of its first element, it is like pass by address, a special case of pass by value.

    Remember, a duplicated address is still an address. For example, a duplicate of address value "main street 123" is still an address, we can locate the element residing this duplicated address and may change that element if necessary.

    Hence, when an array is passed as a parameter, its contents can be changed.

16. Use-defined class and object will not be tested in the midterm. However, existing class like string will be tested.

17. Old midterms and their answers are posted in blackboard.