Name: Inho Choi **ID:** 1801787

True/False: Circle one

1. True / [✓ False] Variable and functions identifiers can only begin with alphabet and digit.

2. True / [✓ False] Compile time array sizes can be non-constant variables.

3. [✓ True] / False Compile time array sizes must be known at compile time.

4. True / [✓ False] An array can have size of 0.

5. True / [✓ False] int m = 15.6; won't compile because the types do not match.

6. [✓ True] / False int n = 11/5; will allocate memory for n and initialize with the value 2.

7. True / [✓ False] Array indices begin with the number 1.

8. [✓ True] / False You can define multiple functions with the same identifier.

9. True / [✓ False] Variables declared local to a function are still accessible after the function completes execution.

10. True / [✓ False] structs by default have private access specification.

11. True / [✓ False] Not defining a destructor for your class will result in a compiler error.

12. True / [✓ False] The following class definition for A requires the entire class definition for B in order to compile.

```
class A {
public:
        void foo( B b);
private:
        B* pb;
};
```

13. True / [✓ False] Declaring a pointer to an object will call the constructor for that object.

Santa Monica College CS 20A: Data Structures with C++ Assignment 1

Name: Inho Choi **ID:** 1801787

Short Answer:

1. Suppose we have a project where Apple is defined in apple.h and Banana is defined in banana.h. Given the main.cpp what can you say for certain about banana.h if this code compiles?

```
#include"banana.h"
#include"banana.h"
int main() {
      Banana b;
      Apple a;
      return 0;
}
```

If this code complies, then we have to write include guard to prevent duplicate header. I wrote

banana.h file

```
#ifndef Banana_H
#define Banana_H
#include "apple.h"

class Banana{
      ⋮
};

#endif
```

Apple.h

```
#ifndef Apple_H
#define. Apple_H

class Apple.{
      ⋮
};

#endif.
```

2. Consider the following class definition for the object S. S maintains an integer id and an integer array and the size of the current number of elements stored.

```
class S {
public:
      S();
      S(int id, int size, int num);
      bool insert(int num);
      void makeCopy(const S &other);
      //...
private:
      int m_id;
      int m_size;
      int m_unique[100];
};
```

Implement the member functions described below.

Name: Inho choi **ID:** 18017897.

a. Implement the default constructor by setting the objects id to -1, size to zero, and each element in the array to zeros. This is implemented outside the class.

```
S::S(){
m_id = -1;
m_size=0;
for(int i=0; i<100; i++ ){
        m_unique[i] = 0;
  }
}
```

b. Implement the three-parameter constructor by setting the object's id and size to the given values. Assign the given num to the elements of the array up to the given size, the rest should be assigned zero. You may assume that size will be greater than or equal to 0 and less than 100. This is implemented outside the class.

```
S::S(int id, int size, int num){
     m_id = id;
    m_size=size;
for(int i=0; i<100; i++){
        if(i<size)
        m_unique[i] = num;
        else
        {
           m_unique[i] = 0;
        }
  }
}
```

Name: Inho Choi **ID:** 1801787

c. Implement the insert function where the given num is placed into the next available slot in the array. insert returns true if the insertion was successful and returns false if there is no space to insert. This is done outside the class.

```
bool s::insert(int num){
    m_size++;
    if(m_size <100){
     m_unique[m_size+1]= num;
         return true;
    }
    else{
       return false;
    }
}
```

d. Implement the makeCopy function that takes as input another S object. makeCopy should make the current object identical to the object being passed into the function. This is done outside the class.

```
Void  S::makeCopy(const S &other){

    m_id = other. m_id;
    m_size = other. m_size;
    for(int i=0; i<100; i++){
      m_unique[i] = other. m_unique[i];
    }

}
```

Name: Jiho Choi **ID:** 18017817

3. Suppose you're tasked with fixing a function that does not work as intended. The function is supposed to compare two cstrings and set the count to the number of identical characters. Two characters are identical if they are the same character and are in the same position in the cstring. This function will be case sensitive so the character 'a' is not the same as 'A'. Note that cstrings are just character arrays that have '\0' as their last character, for example

```
char name[7] = "harry";
```

might looks like this in memory:

h	a	r	r	y	\0	

Usage of this function might look like (Note that the **usage is correct** and should not be modified):

```
int count = 0;
compareCstrings("tacocat", "TACOCAT", count); // should set count to 0
compareCstrings("Harry", "Malfoy", count);   // should set count to 1
compareCstrings("SMC","SBCC", count);        // should set count to 2
```

Currently the function definition is:

```
void compareCstrings(const char str1[], const char str2[], int count) {
    &count = 0;
    int index;
    while (str1 != '\0' || str2 != '\0') {
        if (str1 == str2)
            &count++;
        index++;
    }
}
```

Rewrite the function so that it satisfies specification. Try to keep the general form of the original code, you should not have to add or remove any lines of code, just modify the existing ones. There may be issues with one or more of the input parameters. You may not use the dereference operator , *.

```
Void compareCstrings (const char str1[], const char str2[],
                      int &const) {
    count = 0;
    int index=0;
    while(str1[index]!='\0' || str2[index] != '\0') {
        if (str1[index] == str2[index]).
            count ++;
            index ++;
        }
    }
}
```