OBJECT ORIENTED PROGRAMMING
Written exam

Allotted time: 2h 30 min

1. Indicate the stack allocated variables and the heap allocated variables from the following code snippet.

```
int x = 0;
int main(){
        int a = 10;
        int *pa = &a;
        float *v = new float{3};
        float **pv = &v;
        return 0;
}
```

Stack allocated variables: ________________________________

Heap allocated variables: ________________________________

2. Explain the concept of a dangling pointer. Discuss the dangling pointer problems that might occur in the following code snippet. Don't make any assumptions about the preconditions of the function (i.e. the parameters might point to unallocated memory locations).

```
float *buggy(float *a, float *b, float *c = nullptr){
   float max = *a;
   if (max < *b){
      max = *b;
   }
   if(c && max < *c){
      max = *c;
   }
   return &max;
}
```

3. It is well known that the primitive data types have a limited range of values. For example, the range of values for a *long long int* is typically from -2^{63} to $2^{63}-1$. If we need to work with really really big numbers, we cannot use primitive data types.

   a. Write the **declaration** of a class called BigInteger to represent a large integer. The class should have as members at least: a **heap allocated** array of digits and a variable to indicate the sign of the integer. **Do not implement any other getters, setters, stream operators etc. and any other methods/operators that are not explicitly specified.**

   b. Implement a default constructor and a parameterized constructor for your class. The default constructor creates an array of size 100 and initializes it with zeros. The parameterized constructor takes as input a string (std::string) of characters in the interval [0, 9] and stores the digits in the heap allocated array.

   c. Explain how you used the rule of three in the implementation of this class.

   d. Overload the post-increment (x++) and pre-increment operators (++x) for this class.

      BigInteger& operator++(); // this is the pre-increment operator
      BigInteger operator++(int); // this is the post-increment operator; the int parameter is a dummy parameter, meaning that you won't use it in the implementation, it is just used to distinguish between the pre-increment (++x) and post-increment (x++)

      Both these operators should increment the big integer by 1 (**take into consideration the sign of the big integer when implementing this!**). You can assume that by incrementing the number won't overflow (i.e. you don't need to resize the array).
      The pre-increment operator should return a reference to the current object **after** it is incremented.
      The post-increment operator should return the value of the object **before** it was incremented.

   e. Given the following code snippet:

      ```
      BigInteger i;            // Line 1
      BigInteger i2 = i;       // Line 2
      BigInteger* i3 = &i;     // Line 3
      BigInteger i4;           // Line 4
      i4 = i;                  // Line 5
      BigInteger& i5 = i;      // Line 6
      ```

      Specify on which lines is the assignment operator called and on which lines is the copy constructor called.

4. Given the following code snippet:

```
class Base
{
  int m_a;
protected:
  int m_c, m_b;
public:
  int m_d;

public:
  Base(){
     cout<<"Base ctor"<<endl;
  }
  virtual ~Base(){
     cout<<"Base dtor"<<endl;
  }
  void setA(int i){
     m_a = i;
  }

  int getA(){
     return m_a;
  }

  virtual int getB(){
     return m_b;
  }
};
```

```
class Derived: private Base
{
public:
  Derived(){
     cout<<"Derived ctor"<<endl;
  }
  ~Derived(){
     cout<<"Derived dtor"<<endl;
  }

  void isOK(){
     m_c  = 10;  // Line 1 ✓
     m_d = 10;   // Line 2 ✓
     m_a = 10;   // Line 3 ✓
  }

  int getB(){
     return m_b;  // Line 4 ✓
  }
};

int main(int argc, char const *argv[])
{
  {
     Derived d1;
     d1.getA(); // Line 5 ✓
  }
  cout<<"Done!"<<endl;
  return 0;
}
```

a. For the highlighted lines (Line 1, 2, 3, 4, 5) indicate if the derived objects can access the member of the Base class. Explain why.

b. What is the order of construction and the order of destruction in the context of inheritance? Assuming that you comment out the lines that would give a compilation error, explain this on the current class hierarchy (in which order are the constructors called?, when are the destructors called? and in which order?). What will the program display?

5. What is the output of the following code snippet? Explain why!

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

class Base{
public:
  void method1(){
     cout<<"Base::method1"<<endl;
  }
  virtual void method2(){
     cout<<"Base::method2"<<endl;
  }
  void method3(){
     method1();
     method2();
  }

};

class Derived: public Base{
public:
  void method1(){
     cout<<"Derived::method1"<<endl;
  }
  virtual void method2(){
     cout<<"Derived::method2"<<endl;
  }
};
```

```
int main(){

  Base b;
  b.method1();

  Derived d;
  Base &r1 = d;
  r1.method3();

  Base r2 = d;
  r2.method3();
  return 0;
}
```

6. Lambdas.

   a. What is the purpose of the capture clause in a lambda expression in C++?

   b. Given the following code snippet:

```
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;
int main(){
  vector<int> v = {10, -30, 5, 70, 100, -9, 13};
  int k = 0;
  cout<<"k=";
  cin>>k;
  auto lessThanK = count_if(v.begin(), v.end(),
                  YOUR LAMBDA HERE
);

  cout<<"Number of elements less than "<<k<<": "<<lessThanK<<endl;
  return 0;

}
```

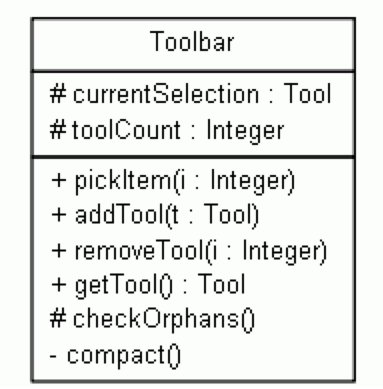
Write the lambda expression for the *count_if* function, such that *lessThanK* will store the number of elements that are less than the value *k* inputted by the user. The *count_if* function counts the elements that comply with the condition specified by the lambda. The lambda takes as parameter a single integer (an element from the array) and returns true if that parameter is less than *k*, and false otherwise.

~~7.~~ What is the output of the following program?

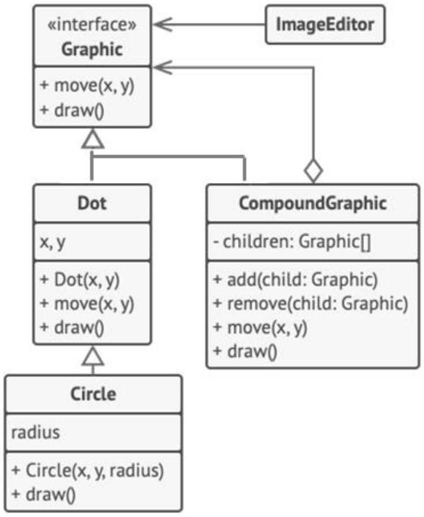
```
#include <math.h>
#include <utility>
float divide(float a, float b){
  if(b == 0)
     throw '0';
  float d = static_cast<float>(a)/b;
  return d;
}
float ssqrt(float v){
  if(v < 0){
     throw -1;
  }
  return sqrt(v);
}
                              2        1        5
std::pair<float, float> solve(float a, float b, float c){
  std::pair<float, float> res{0, 0};
  float delta = static_cast<float>(b*b - 4*a*c);
  try {
     res.first = divide(-b + ssqrt(delta), 2*a);
     res.first = divide(-b - ssqrt(delta), 2*a);
  } catch (int &i) {
     cout<<"Caught an integer exception "<<i<<endl;
     res.first = 0;
     res.second = 0;
  }
  return res;
}

int main(){
  cout<<"Equation 1: "<<endl;
  try {
     std::pair<float, float> res1 = solve(0, 2, 15);
     cout<<"The results of the first equation are: "<<res1.first<<";
"<<res1.second<<endl;
  } catch (...) {
     cout<<"Caught an exception of any type!"<<endl;
  }
  cout<<"Equation 2: "<<endl;
  std::pair<float, float> res2 = solve(2, 1, 10);
  cout<<"The results of the second equation are: "<<res2.first<<";
"<<res2.second<<endl;
  return 0;
}
```

8. Class diagrams I. Write the **declaration** of the following class. Pay attention to the visibility of each member! You can assume that the Tool class is already defined in a file "Tool.h". You can use *int* for Integer.



9. Class diagrams II.
   a. Name the relationships between the Graphic and CompoundGraphic classes. How are these implemented in code?
   b. What is the relationship between the ImageEditor and Graphic class? How would this relationship be implemented in code?
   c. Name the design pattern depicted in the class diagram below.



10. Standard library (STL) containers. Explain the difference between a STL array and a STL vector.

Declare a STL array and store into it the days of the week.

11. Write a templated class to represent a Pair. This class should have two **public** (yeah, I know!) members *first* and *second* of **different** data types.

    Instantiate two Pair objects: the first one has members of the types int and float, while the second one has the members of type string and int.

12. **Extra.** Name two computer scientists who had a contribution to the C/C++ programming languages or to the development of object-oriented programming paradigm.

RUBRIC

Item	**Grading/description**
Default	**1p**
Exercise 3	**3p**
Exercise 1, 2, 4, 5, 6, 7, 8, 9, 10, 11	**each 0.6 p**
Exercise 12	**+0.25 (extra)**

BEST OF LUCK!