2020/11/30
CS Department
Dr. Klaus Baer

Bar Code

Programming III,
Fall 2020
Mid Term Exam

Instructions: **Read Carefully Before Proceeding.**

1- No calculators, book or other aids are permitted for this test.
2- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem.
3- Attempt as much of the problems as you can within the time limits. The more you solve the higher your score is expected to be.
4- Read all the problems carefully before starting, and start with the easiest question you find.
5- This exam booklet contains 16 pages, including this one. *Extra sheets of scratch paper are attached* and have to be kept attached.
6- When you are told that time is up, stop working on the test.
7- Total time allowed for this exam is ***120 min.***

Good Luck!

Question Number	1	2	3	4	5	6	-	-	-	-	Total
Maximum Score	12	28	10	16	8	26					100
Obtained Score											

Task 1 (12 Punkte = 5 + 4 + 3)

a) The following program fragment shall be given. Indicate which assignments lead to a compile error and justify your answer.

```
int main() {
        int i1 = 42;
        int * p1 = &i1;
        const int * p2 = &i1;
        int * const p3 = &i1;
        const int * const p4 = &i1;
```

	Valid?	Justification
`*p1 = *p2;`	ok	writing forbidden because pointer to constant
`p1 = i1;`	no	Assignment of int to int* not possible
`p3 = p1;`	no	writing forbidden constant pointer
`p2 = p4;`	ok	Data kept constant, redirecting p2 no problem
`p1 = p2;`	No	Conversion from 'const char *' to 'char *' not possible.

```
}
```

b) Explain the various meanings of `const` in C++:

- `const int i = 42;`

  declaring a constant value

- declared as member of a class: `void member_method() const {}`

  method member_method will not change the status of the object it is called for. It can be called within foo in the last point: p.member_foo() does not change p.

- `void global_method(const MyClass &p){}`

  method global_method promises to its users that it will not change the value of reference p. Similar to call-by value without copying.

- `const MyClass & a_method(){}`

  a_method () returns a const reference to MyClass. Via this reference, the value may not be changed as it is declared const

c) What are “member initializer list expressions”?
   Explain and give an example.

   In the definition of a constructor of a class, *member initializer list* specifies the initializers for direct and virtual bases and non-static data members.

   Before the compound statement that forms the function body of the constructor begins executing, initialization of all direct bases, virtual bases, and non-static data members is finished. Member initializer list is the place where non-default initialization of these objects can be specified.

Task 2 (28 Points = 14+3+2+5+4)

The following class definition shall be given:

```cpp
class P {
public:
     P()               { cout << " cP"; }
     P(const P& a)     { cout << " copyP"; }
     ~P()              { cout << " ~P"; }
};

class A {
private:
    P* pP=0;
public:
    A()                { cout << " cA"; }
    A(const A& a)      { cout << " copyA"; }
    virtual ~A()       { cout << " ~A"; }
    void method(A a){ cout << " methodA"; }
};

class B : public A {
private:
    P p;
public:
    B()                { cout << " cB"; }
    B(const B& a)      { cout << " copyB"; }
    ~B()               { cout << " ~B"; }
    void method(A a){ cout << " methodB";a.method(a); }
};
```

a) For the test () function below, after each line, specify which outputs appears on the console during evaluation.
Please write the word "NOTHING" in cases where you want to express that no output is generated. Leaving a field blank means that you have not provided an answer and therefore will not get points for it.

Nr.	Code	Output
1	`void test(){`	(Nothing) no grading
2	`B b1;`	cA cP cB
3	`B b2 = b1;`	cA cP copyB
4	`A a1;`	cA
5	`A a2;`	cA
6	`A* pa1 = &a2;`	Nothing
7	`A* pa2 = new A();`	cA
8	`B* pb3 = new B();`	cA cP cB
9	`a2 = a1;`	Nothing
10	`pa2→method(b1);`	copyA methodA ~A
11	`pb3→method(b2);`	copyA methodB copyA methodA ~A ~A // 2 P
12	`delete pa2;`	~A
13	`}`	~A ~A ~B ~P ~A ~B ~P ~A // 2 P

b) The destructor of class A is declared virtual.

- What does this declaration mean?
- In which cases this is absolutely necessary for a destructor?
- What does the declaration do in the current case?

Virtual leads to late binding, that is, the system checks at runtime which object type really exists and calls the methods of this object type.

Especially destructors of base classes have to be declared virtual, otherwise memory leaks can occur when deleting via base class pointers, if derived classes have requested memory in the constructor.

Correct in the current case, but since no memory is requested in derived classes, no effect.

c) Which function is called in line 9 of the test method and what does it do?

Default assignment operator for class A:
Copies element attributes from object a1 to a2

d) Explain in detail the output in line 11!

- How the parameter is passed to the method? What are the consequences? Which functions are called?

- b2 is passed by value, slicing takes place: b2 becomes an A object
- is created on stack by copy constructor
- method from class B is used, since call via B pointer
- in method the class A method is called and passed again via stack → copyA
- both objects on stack must be deleted ~A~A

e) Now, let us change the declaration of `method` in `A` and `B` as follows:

```
    void method( A& a);
```

What are the changes in the output?

What happens, if the declaration of `method` in `A` and `B` will be changed to

```
    void method(const A& a); ?
```

Explain!

Line 10 and 11 will change to:
methodA
methodB methodA

call-by reference → no copy ctor will be called

Declaring the parameter const works in A but not in B → syntax error
"method" in B calls "method" for a-object, but this is not declared const.
So it is not allowed to call it for a const reference to a.

Task 3 (10 Points = 5 + 5)

a) Please answer the following questions::

	Correct	Wrong
Static data elements are always const		x
A public static data elements may not be accessed via an object		x
Static data elements of a class are also called “class variable”	x	
Static methods of a class may be called only, after an object of that class has been created.		x
Static methods may only use static data elements	x	

b) What is a **namespace** in C++?
Explain the meaning and purpose of namespace and use a syntax example to explain how to use namespaces.

Namespaces provide a method for preventing name conflicts in large projects.
Symbols declared inside a namespace block are placed in a named scope that prevents them from being mistaken for identically-named symbols in other scopes.
Multiple namespace blocks with the same name are allowed. All declarations within those blocks are declared in the named scope.

```
namespace Geo {
        class GeoObject{
                ...
        };
} // end namespace
```

Or
Using namespace std ;

Task 4 (16 Punkte= 10 + 6)

A circular list is a linear list of elements. It is circular, because the `next` pointer of the last list element points to the head of the list. You could think of a ring.

The following class represents such a ring:

```
class Ring {
private:
      struct ListElement{
            int value;
            ListElement* next;
      };
      ListElement* head;
      int size;
public:
      Ring( int n=0 );
      ~Ring();
      void print();
      // ... other methods
};
```

a) For the class Ring, implement the corresponding constructor that creates a circular list with n list elements.

- For n > 0, `head` should point to an element with value 1 and the following elements shall have ascending values 2, 3, ..., n, respectively.
- For n <= 0, an empty ring object shall be generated.

```
Ring::Ring( int n ) {
// if n <= 0
size = n;
head = NULL;
// otherwise
if ( n > 0 ) {
        // special treatment of 1. element
        ListElement* t = new ListElement;
        t->value = 1;
        head = t;
        // all others
        for ( int i=2; i <= n; ++i ){
                t->next = new ListElement;
                t = t->next;
                t->value = i;
        }
        // close the ring
        t->next = head;
}
}
```

b) Develop the `destructor` for `Ring:`

```
Ring::~Ring(){
for(int i =0; i<size; ++i){
        ListElement* tmp = head->next;
        delete head;
        head = tmp;
}
```

Task 5 (8 Points)

Explain what is meant by a downcast.

- Describe the situations in which downcasts are problematic.
- Describe how to deal with them correctly.
- Give an example.

Base class pointer is casted in pointer to derived class
save: Base class pointer actually points to the type of the cast
unsafe: At runtime, the base class pointer points to the parent object of teh cast type

Correct handling:
dynamic-Cast returns NULL pointer if type does not match,
This can be checked and, if necessary, error handling can take place.

```
Kfz * kfzPtr = new Kfz;
Passenger car * pkwPtr = NULL;
pkwPtr = dynamic_cast<Pkw*>(kfzPtr); // returns zero pointer!
if (pkwPtr == 0)
cout << "Error!!" <<< endless
else {
pkwPtr->display();
pkwPtr->get in(3);
}
```

Task 6 (26 Points = 8+3+1+4+4+6)

A set is a collection of elements (=values) in which each element may be contained at most once. A class `DblSet` is to be implemented, which manages a set of double values.

For this purpose, the class provides the following functionalities:

- Checking whether a value is already included in the set.
- Insert a new value
- Determining the number of values contained in the set.

The elements of the set are stored in a dynamically created array. If necessary, the size of the array will be increased automatically. If an object of this class is created, the set is initially empty (i.e. does not contain any values), but memory for `INITIAL_SIZE` many elements should be reserved already.

`INITIAL_SIZE` is supposed to be a static data element of the class `DblSet`.

On insertion, new elements are placed after of the other elements in the array. If the value is already included in the set, the set remains unchanged. Otherwise, the value is stored at the end of the occupied range in the array. If the array is fully occupied, the capacity of the array will be doubled.

The method IsIncluded() checks whether a specified value is included in the set.

The size() method returns the number of values currently contained in the set.

Example:
The following piece of code,

```
DblSet s1;

cout << "s1: " << s1.size() << " elements" << endl;
s1.insert(3.1);
s1.insert(2.8);
s1.insert(0.01);
s1.insert(3.1);
cout << "s1: " << s1.size() << " elements" << endl;

if (s1.isIncluded(2.8)){
     cout << "2.8 is included within s1" << endl;
}
```

along with your implementation of the DblSet class, should produce the following output:

```
s1: 0 elements
s1: 3 elements
2.8 is included within s1
```

a) Declare the class `DblSet` in the header file "`DblSet.h`". To do this, first consider,

- which data elements you need,
- consider a reasonable assignment of the access rights and
- define the interfaces of the required methods appropriately.

Declare the method `size()` implicitly inline.

```
class DblSet {                          1 for correct public & private
public:
DblSet();                               0,5
DblSet(const DblSet& other);            1
~DblSet();                              0,5
bool isIncluded(double wert);           1
void insert(double wert);               1
int size() {
        return noOfElements;            1 implicit inline
}
private:
double * values;                        1 for all attributes
int capacity;
int noOfElements;
static const int INITIAL_SIZE =2;   1 for class constant with initialization
};
```

b) Implement the default constructor for the DblSet class.

```
DblSet::DblSet() {
values = new double[INITIAL_SIZE];
capacity = INITIAL_SIZE;
noOfElements = 0;
}
```

c) Implement the destructor for the DblSet class.

```
DblSet::~DblSet() {
delete[] values;
}
```

d) Implement the `IsIncluded()` method, which checks whether the specified value is included in the set.

```
bool DblSet::isIncluded(double value) {
const double THRESHOLD =  0.000000001;
for (int i = 0; i < noOfElements; ++i) {
        if (abs(values[i] - value)/values[i] <= THRESHOLD){  // 2p for comp.
                return true; //Wert gefunden
        }
}
return false; //Wert nicht gefunden
}
```

e) Implement the insert method that inserts the specified value into the set if not already included and that handles the memory management (→ doubling when full)

```
void DblSet::insert(double value) {
if (! isIncluded(value)) {
        if (noOfElements == capacity) {
                capacity *= 2;
                double * newValues = new double[capacity];
                for (int i = 0; i < noOfElements; ++i){
                        newValues[i] = values[i];
                }
                delete[] values;
                values = newValues;
        }
        values[noOflements ++] = value;

}
}
```

f) Implement the copy constructor and explain why the copy constructor is necessary in this case.

In the constructor memory is reserved dynamically for an array. It must be ensured in the copy constructor that memory is also reserved in the copied object. Otherwise, copy and original would work on the same array and mutually overwrite each other's data, since the default constructor generated by the system would only create a so-called "flat copy", i.e., only a copy of the pointer. 3p

```
DblSet::DblSet(const DblSet& other) {
if(this != &other){
        capacity = other.capacity;
        delete values;
        values = new double[capacity];

        for (int i =0; i < other.noOfElements; ++i){
                values[i] = other.values[i];
        }
}
}3p
```