İ.T.Ü.
Computer Engineering Department
Dr. Feza BUZLUCA
Çiçek ÇAVDAR

Object Oriented Programming 1st Midterm Examination
SOLUTIONS

Answer 1:

a)

Message:	Reason:
Start	cout in main
Function 1	Default constructor for a
Function 1	Default constructor for b
Function 1	Default constructor for c
Operation	cout in main
Function 2	c is copied into stack as parameter for operator+
	A new object is created in stack and copy constructor is invoked to c into new object
Function 4	cout in opertor+
Function 1	constructor for temp
Function 2	temp is copied into stack as return value
Function 5	destructor of temp
Function 5	destructor of parameter in stack (copy of c)
Function 3	assignment operator of a
Function 5	destructor of return value in stack
End	cout in main
Function 5	destructor of c
Function 5	destructor of b
Function 5	destructor of a

b)

```
#include <iostream>
using namespace std;
class Aclass{
  private:
    int i;
  public:
    /* return statement in operator+ needs a constructor with
       one argument */
    Aclass(int i_in=0){cout <<"Function 1" << endl; i=i_in;}
    Aclass operator+(Aclass &);
};

Aclass Aclass::operator+(Aclass &in_c)
{
  cout << "Function 4" << endl;
  return Aclass(i+in_c.i);
}
```

Copy constructor and assignment operator are unnecessary, because the default functions built by the compiler do the same thing.
The destructor is also useless. It ‘destructs’ nothing.

c)

Message:	Reason:
Start	cout in main
Function 1	Constructor of a
Function 1	Constructor of b
Function 1	Constructor of c
Operation	cout in main
Function 4	cout in operator +
Function 1	Constructor to create return object in stack
End	cout in main

Answer 2:

a)

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

class Array{
  int size, avail;
  int *contents;
 public:
  Array(int);                           // constructor
  Array(const Array &);                 // copy constructor
  Array& operator=(const Array &);      // assignment operator
  int operator<(const Array &) const;
  bool write(int,int);
  int read(int);
  void empty() {avail=size;}
  ~Array(){delete contents;}            //destructor
};
/*** Constructor: Takes the size of the array ****/
Array::Array(int n)
{
  contents = new int[n];
  size=avail=n;
}
/*** Copy Constructor: Copies the contents of an array into a
new created object ***/
Array::Array(const Array &in_array)   // Copy Constructor
{
  size = in_array.size;
  avail=in_array.avail;
  contents = new int[size];
  for (int i=0; i<size; i++)
      contents[i]=in_array.contents[i];
}
/*** Assignment Operator: Copies the contents of an array into
an existing object ***/
Array& Array::operator=(const Array &in_array)//Assignment op.
{
   size = in_array.size;
   avail = in_array.avail;
   delete [] contents;                  // delete old contents
   contents = new int[size];
   for (int i=0; i<size; i++)
```

```
        contents[i]=in_array.contents[i];
    return *this;
}
/*** operator<: Compares available spaces of two arrays ***/
int Array::operator<(const Array &comp_array) const
{
  return avail-comp_array.avail;
}

/*** write: Writes an element into a given position. If the
operation is successful it returns true, otherwise false ***/
bool Array::write(int value,int position)
{
  if(position>=0 && position<size && avail>0)
  {
     contents[position]=value;
     avail--;
     return true;
  }
  else return false;       // WRITE ERROR
}

/*** read: Returns the value of the element in the given
position. If an error occurs, it returns -1 ***/
int Array::read(int position)
{
  if(position>=0 && position<size)
  {
     avail++;
     return contents[position];
  }
  else  return -1; // READ ERROR
}
```

b)

```
/***** Stack Class *********/
class Stack{
  Array values; //includes an Array object to hold pushed data
  int stackpointer;
  int ssize;
 public:
  Stack(int);                          //constructor.
  Stack(const Stack &);                //copy constructor
  ~Stack(){;}                          //destructor
  Stack& operator=(const Stack &);   //assignment op.
  int operator<(const Stack &) const ;
  bool push(int);
  int pop();
  void emptystack();
};

/*** Constructor: creates a stack of given size. It calls the
constructor of Array ***/
Stack::Stack(int n):values(n)
```

```
{
   stackpointer=0;
   ssize=n;
}
/*** Copy Constructor: Copies the contents of a stack into a
new created stack. It calls the copy constructor of Array ***/
Stack::Stack(const Stack &in_stack):values(in_stack.values)
{
   stackpointer=in_stack.stackpointer;
}

/*** Assignment Operator: Copies the contents of a stack into
another existing stack. It calls the assignment operator of
Array ***/
Stack& Stack::operator=(const Stack &in_object)
{
  values=in_object.values;   //calls the assign. Op. Of Array
  stackpointer=in_object.stackpointer;
  return *this;
}

int Stack::operator<(const Stack &comp_stack) const
{
   return values<comp_stack.values;
}

/*** push: Pushes a value into stack. If there is an error it
returns false, otherwise true ***/
bool Stack::push(int value)
{
  if (stackpointer<ssize)
  {
     values.write(value,stackpointer);
     stackpointer++;
     return true;
  }
  else return false;
}

/*** pop: Returns(pop) a value from stack. If there is an
error it returns 0 ***/
int Stack::pop()
{
  if (stackpointer>0)
      return values.read(--stackpointer);
  else
      return 0;
}

/*** emptystack: All data is discarded ***/

void Stack::emptystack()
{
  values.empty();
  stackpointer=0;
}
```

c)

```
void main()
{
  Stack s1(15), s2(15); // Two stacks of size 15 are created.
  s1.push(2);               // Push some values
  s1.push(4);
  s2.push(5);
  s2.push(3);
  s2.push(4);
  s1.push(12);
  s1.push(44);
  int c=s1<s2;           // Compare stacks
  Stack *sp;             // A pointer to Stack
  if (c==0){             // Stacks are equal size or empty
     if(s1.pop() && s2.pop()) //is there any element in stacks
       cout<<"Stacks have the same size"<<endl;
     else
       cout<<"Stacks are empty"<<endl;
  }
  else{    //c is not zero (one of the stacks has less space)
     if (c<0){
        cout<<"Empty spaces of s1 is less than s2"<<endl;
        sp=&s1;      // sp points to s1
     }
     else if(c>0){
        cout<<"Empty spaces of s2 is less than s1"<<endl;
        sp=&s2;     //sp points to s2
     }
     int i=sp->pop();  //pop elements from stack pointed by sp
     while (i!=0){
        cout<<i<<" ";
        i=sp->pop();
     }   // end while
  } // end else
}
```