CSC 112 MIDTERM EXAM
FALL 2003

You will have 110 minutes to complete this exam. All work should be written in the exam booklet. Start with the questions that you know how to do, and try not to spend too long on any one question. Partial credit will be granted where appropriate. Good luck!

1. Argument Passing (12 points). Predict the terminal output of the following program:

```
#include <iostream>
using namespace std;
const int d = 1;
int myfun(int a, int &b,
int *c) {
  a = a+*c;
  b = a-b;
  *c = b+*c;
  cout << a << endl;
  cout << b << endl;
  cout << *c << endl;
  cout << d << endl;
  return (b+*c-a);
}
int main() {
  int a = -1, b = 2, c =
3, d = 5;
  a = myfun(b,c,&d);
  cout << a << endl;
  cout << b << endl;
  cout << c << endl;
  cout << d << endl;
}
```

Result:

```
7
4
9
1
6
2
4
9
```

2. Arrays (12 points). Predict the terminal output of the following program:

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

//*******************************************
int main() {
  int a[6] = {4, 1, 3, -7, 1, 0};
  int minval = a[0], minpos = 0;
  for (int i = 0; i < 6; i++) {
    if (a[i] < minval) {
      minval = a[i];
      minpos = i;
      cout << "Value:  " << minval << endl;
    }
    cout << "Position:  " << minpos << endl;
  }
}
```

Result:

```
Position:  0
Value:  1
Position:  1
Position:  1
Value:  -7
Position:  3
Position:  3
Position:  3
```

3. Classes (20 points). You are writing the software to control a jukebox. It will store up to 20 songs waiting to be played, using a statically allocated array. New songs to be played can be added to the end of the array via an `enterSong()` method. Periodically, another piece of software calls the `nextSong()` method to find out what to play next, removing the first song in the array and moving the others downwards. The `numSongsWaiting()` method reveals how many songs are currently waiting.

a. Assuming that class `Song` is defined in the file `Song.h`, give a complete definition (but not implementation) of class `Jukebox`, including required constructors, destructor, the methods mentioned above, and any properties needed to support these operations. Your answer should demonstrate good coding practices as taught in class.

```
#ifndef _JUKEBOX_H_
#define _JUKEBOX_H_
#include "Song.h"
const int MAX_SONGS = 20;
class Jukebox {
 public:
  Jukebox();
  Jukebox(const Jukebox&);
  ~Jukebox();

  int numSongsWaiting() const;

  void enterSong(Song s);
  Song nextSong();

 private:
  Song songs[MAX_SONGS];
  int songsWaiting;
};
#endif
```

b. Write an implementation for the accessor `numSongsWaiting()`.

```
int Jukebox::numSongsWaiting() const {
  return songsWaiting;
}
```

c. Write an implementation for the copy constructor of the class described above, as it would appear in `Jukebox.cpp`. Use at least one initializer in your implementation.

```
Jukebox::Jukebox(const Jukebox &jb):
  songsWaiting(jb.songsWaiting)
{
  for (int i = 0; i < songsWaiting; i++) {
    songs[i] = jb.songs[i];
  }
}
```

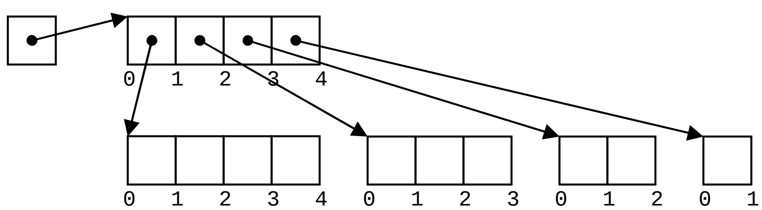
d. Write an implementation for the `enterSong()` method described above, as it would appear in `Jukebox.cpp`. If there are already 20 songs waiting, it should do nothing.

```
void Jukebox::enterSong(Song s) {
  if (songsWaiting < MAX_SONGS) {
    songs[songsWaiting] = s;
    songsWaiting++;
  }
}
```

4. Array Allocation (16 points). Below is a fragment of code allocating some memory.

```
int **tri = new (int*)[4];
for (int i = 0; i < 4; i++) {
  tri[i] = new int[4-i];
}
```

a. Draw the data structures in memory that would be created by this code. Show the actual number of boxes in each array allocation, and label the indices.



b. Write another code fragment that would properly deallocate the dynamic memory allocated above.

```
for (int i = 0; i < 4; i++) {
  delete[] tri[i];
}
delete tri;
```

5. Style (8 points). Give two reasons why keywords like **`const`** and **`static`** should be used whenever possible.

Using such keywords documents the manner in which variables in your program are used. They allow the compiler to check for you that the variables are being used consistently with their intent. They can also allow the compiler to create a more efficient program in some cases.

6. **Inheritance** (16 points). Consider the class definitions given below.

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

class Pet {
public:
  Pet();
  Pet(string n, int a);
  Pet(const Pet&);
  ~Pet();

  string getName() const;
  int getAge() const;
  void setName(string n);
  void setAge(int a);
  void printDescription();

private:
  string name;
  int age;
};

class Dog {
public:
  Dog();
  Dog(const Dog&);
  ~Dog();

  string getName() const;
  int getAge() const;
  int getLicense() const;
  void setName(string n);
  void setAge(int a);
  void setLicense(int ln);
  void printDescription();

private:
  string name;
  int age;
  int license;
};
```

a. Suppose that class `Dog` is going to be rewritten as a subclass of class `Pet`. What changes would have to be made to the class `Dog` definition? Eliminate any unnecessary code.

```
class Dog : public Pet {
public:
  Dog();
  Dog(const Dog&);
  ~Dog();

  int getLicense() const;
  void setLicense(int ln);

private:
  int license;
};
```

b. Suppose that the implementation of `Dog::printDescription()` includes displaying the license number of the dog. You have a number of different pets accessible through an array of `Pet` pointers, and you plan to write a loop that will call `printDescription()` for each of them. How can you make the license number print out for all the pets who are dogs? Be specific about what you would change and where.

Declare `Pet::printDescription()` ***to be*** `virtual`***. Return the declaration of*** `void printDescription()` ***to the definition of class*** `Dog`***, and in the implementation, make it call*** `Pet::printDescription()` ***before printing the license number.***

7. Sorting (10 points). The array shown below is to be sorted into increasing order using selection sort. Draw a diagram of the array after each swap in the selection sort algorithm. Assume that the sorted region is grown from the left side of the array.

8	3	2	9	5
2	3	8	9	5
2	3	8	9	5
2	3	5	9	8
2	3	5	8	9
2	3	5	8	9

8. Program Complexity (6 points). You're planning a scientific data-processing program that will perform weather simulations using data from ***n*** weather stations scattered all over the world. The number ***n*** is large, and is expected to continue to grow as more stations are added. You have a choice of different algorithms, with different asymptotic complexities. Rank these choices in order from best to worst.

a. $O(3n^2)$ -- ***Second best***

b. $O(1000n)$ -- ***Best***

c. $O(1+1.1^n)$ -- ***Worst***