



The University of
Chicago
Department of
Computer Science

CMSC 15200 – Introduction to Computer Science 2
Summer Quarter 2005
Optional take-home midterm (08/12/2005)
Due: 08/14/2005 @ 5pm

Name:

Student ID:

Instructor:

Borja Sotomayor

Do not write in this area						
1	2	3	4	5	6	TOTAL
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Maximum possible points: 115						
2-hour midterm: <input type="checkbox"/> Very bad <input type="checkbox"/> Bad <input type="checkbox"/> Average <input type="checkbox"/> Good <input type="checkbox"/> Very good						

Instructions

***** READ THESE INSTRUCTIONS CAREFULLY BEFORE PROCEEDING *****

Exam Format

This is a programming exam. You will be provided with a set of C/C++ files and will be asked to add new functionality or modify existing code. This exam is meant to be done entirely on a computer. Although there are no theory questions, you will need a clear understanding of C/C++ programming, data structures, and algorithms to understand and solve the exercises in this exam.

You must hand in all your files at the end of this exam using the hws submit application used to hand in the homework assignments and labs. You must also hand in this handout. However, do not write anything in the handout as it will not be read by the grader. If you need to include comments regarding your solution to a problem, do so directly in the code using C/C++ comments.

Do not worry if your exam stops working (i.e. does not compile or produces wrong results) right before you hand in the exam. Small programming errors, specially in the tense final minutes of an exam, are understandable and will rarely result in a point deduction (as long as it is a small error which the grader can correct easily). On the other hand, handing in a program with multiple and gross compiler errors will result in considerable point deductions (or will not be graded at all). Remember: the grader will grade the exam taking into account that you had access to documentation, books, websites, etc. and (more importantly) a compiler during the course of this exam.

Two submissions

Although you have approximately 48 hours to do this take-home midterm, this is *not* an exam that should take you that long to do! This exam should give you an idea of what



you can expect in the final exam. Therefore, we encourage you to hand in two versions of your midterm:

- First of all, do the midterm in the same conditions as the final exam (reserving two hours to do the midterm). At the end of the two hours, hand in all your files. Include an empty file called 2HOUR.TXT with your files so the instructor will know what submission this is. This submission will *not* be graded, but the instructor will review it and will rate the exam on a scale from “Very bad” to “Very good” (see grading box above). This will give you an idea of how you would have done if this were the final exam.
- Next, you can use the rest of your time to go over the midterm, finish any exercises you did not complete during the two hours, test your code more thoroughly, etc. This second submission *will* be graded and will account for 10% of your final grade.

Rules

- You are allowed to use all the resources permitted during the final exam: books, notes, websites, homework, labs, etc.
- You must do this midterm by yourself. You are *not* to discuss this midterm with any of your classmates.
- The usual rules on academic honesty apply: no plagiarizing and no using code from books/websites/etc. without citing your sources.
- Late submissions will **not** be accepted *under any circumstances whatsoever*. If you feel you will not be able to work on the midterm during the weekend, then inform the instructor before the end of class on Friday. This will be equivalent to not having taken the optional midterm at all (remember your final exam will then account for 40% of your grade). On the other hand, if you hand in your midterm late, you will get zero points on the midterm.

*By taking this midterm home with you,
you are agreeing to the above rules.*

Midterm files

You can download the midterm files from the course website. Unless otherwise stated (on paper or in the code), you should not modify any of these files.

midterm.h

This header file declares a Client struct which you will use throughout the exam:

```
struct Client {  
    string name;  
    char* account;
```



```
int age;  
float monthlyFee;  
};
```

It also declares a function which is already implemented for you (printClient) and three functions you will have to implement in the exam.

list.h

Header file for a linked list of Client structs you will have to use in the exam.

list.cpp

Implementation of some of the list functions (you will have to implement some of them)

main.cpp

This file contains the main() function. It includes several test calls to the functions you have to implement in this exam. If you correctly implement all the functions in the exam, you will see the output shown in the last page of this handout. Notice how most of the code is commented out. You should uncomment each block of code (corresponding to each of the exercises in the exam) as you finish each exercise, so you can test if each function works correctly.

During this exam, you can modify this file to try out different parameters when calling the functions. However, make sure that the main.cpp file you hand in does not include any additional code you might have added.

midterm.cpp

Finally, this is the file where you must write the implementation of all the additional functions required in this exam. This file also includes the implementation of the printClient() function.

Tips

- Don't wait until the last minute to compile your exam and see if it works. Compile often to check if there are any errors in your code. It is easier to correct a few errors every couple of minutes, than to have to deal with dozens of errors in the final minutes of the exam.
- The exercises are independent of each other. Start by doing the one you feel most comfortable with. If you get stumped on an exercise, comment it out, and move on to a different exercise.
- Don't reinvent the wheel. Some of the exercises are similar to problems you've encountered before in class, homework assignments, and labs. Reuse code whenever possible and leverage the standard libraries in your program.



EXERCISES

These exercises are all independent of each other.
You can solve them in any order.

Exercise 1 <<10 points>>

Implement the following function:

```
void initClient(Client &c);
```

This function will ask the user for the client's data and store it in parameter "c". You must validate that the account has a correct value (an account number has 7 characters, starts with two uppercase letters, and ends with a lowercase letter. There is no restrictions on the three characters in between). If it is not valid, then you will set the account number to "AA000a". You must also validate that the age and the monthly fee are positive integers.

Exercise 2 <<15 points>>

Implement the following function:

```
??? clientsToArrayFloats(???);
```

This function accepts the following as input:

- An array of Client structs of length N . Each position is assumed to have a valid Client struct.

This functions must return the following:

- An array of floats of length N . Each position must contain the monthly fee of the corresponding client in the array of Client structs.
- The sum of all the monthly fees.

The above does not represent the exact parameters and return types the function will have. You must determine how many parameters the function will have, what their types will be, and what the return type will be.

Note: Remember to correctly declare/define the function in both the "midterm.h" file and the "midterm.cpp" file. Otherwise, your program will not compile.



Exercise 3 <<20 points>>

Implement the following function:

```
int writeToDisk(char* filename, Client ca[], int clients);
```

This function will take an array "ca" of Client structs, and write it to file "filename" using the following text format:

```
<name><newline>  
<account><newline>  
<age><newline>  
<monthlyFee><newline>  
...  
...  
...  
<name><newline>  
<account><newline>  
<age><newline>  
<monthlyFee>
```

For example:

```
John Doe  
XY557a  
24  
100.5  
Jane Doe  
PI314e  
72  
200.5
```

The function will then return the number of clients written to disk. This does not mean that you need to return the *clients* parameter. You must keep a count of how many clients have been written to disk and return that value.

Exercise 4 <<25 points>>

Implement the following function:

```
void recursivePrintData(ClientList &l);
```

This function writes the contents of the list to the standard output. It is functionally equivalent to the `printData` function seen in class and included with the list implementation. However, you are asked to implement this function *recursively*. Hint: You will need an extra function.



Exercise 5 <<20 points>>

Implement the following function:

```
void cleanList(ClientList &l);
```

This function will traverse the list and do the following:

- If the age of the client is 65 or greater, reduce his/her monthly fee by 10%.
- If the age of the client is less than 18, remove that client.

Exercise 6 <<25 points>>

Implement the following functions:

```
int readFromDisk(char* filename, ClientList &l);  
void insertInOrder(ClientList &l, Client &c);
```

Function "readFromDisk" must read a client file from disk (file "filename") and place its contents in the provided list. The client file has the same format as the one described in exercise 3. If you have not done exercise 3, a sample file is provided in the midterm files so you can test this function.

When inserting the new Clients into the list, the "readFromDisk" function must use the "insertInOrder" function. As described in class, this function inserts a new node in the list making sure that an ordering is preserved in the list. In this case, the nodes must be in increasing order by name.



EXPECTED OUTPUT

EXERCISE 1

Name: Roger Kingston-Hughes
Account #: XY567a
Age: 24
Monthly fee: 134.70
Roger Kingston-Hughes XY567a 24 134.7

EXERCISE 2

Total fees: 1502.5
Fees: 100.5 200.5 300.5 400.5 500.5

EXERCISE 3

5 clients written to file.

EXERCISE 4

Cornelius Doe	LK874k	35	500.5
Lucius Doe	BD123i	67	400.5
Rufus Doe	DA420h	16	300.5
Jane Doe	PI314e	72	200.5
John Doe	XY557a	24	100.5

EXERCISE 5

Cornelius Doe	LK874k	35	500.5
Lucius Doe	BD123i	67	360.45
Jane Doe	PI314e	72	180.45
John Doe	XY557a	24	100.5

EXERCISE 6

5 clients read from file.

Cornelius Doe	LK874k	35	500.5
Jane Doe	PI314e	72	200.5
John Doe	XY557a	24	100.5
Lucius Doe	BD123i	67	400.5
Rufus Doe	DA420h	16	300.5