

Programming project

You are to design, document, and write a program that manages a collection of *web pages*, implemented as a collection of structures or objects.

1. Each web page has the following attributes:
 - Short ID, such as a number;
 - Name (one word);
 - URL;
 - Dates visited.
2. Your program should execute a loop that displays a menu, prompts for the user to choose an option, and executes the option. The option may be coded as a letter or number. In addition to “Quit,” include the following user options:
 - input one site record;
 - display one web-site record, selected by ID or name;
 - display all sites, sorted on any field;
 - update existing record;
 - delete record;
 - save all records to disk;
 - retrieve all records;
 - search for site by ID or by name.
3. Use a structure type or class to represent one site. Consider using a second structure type or class to represent an entire site list, including an array of site structures. Use member functions or function parameters to share collection data among functions. *Challenge:* implement the collection as a random-access file.
4. The final code should be in at least two separately compiled *.cpp* files. Use appropriate *.h* files for constant and function declarations, and class declarations. Use *.cpp* files for function definitions. Suggestion: put all the function definitions for one class or abstract data type in one *.cpp* file.
5. *Document* your program with a *user manual*, a text description of implementation design, a *class diagram* and at least one *module-hierarchy chart*. Document each function by stating its purpose.
6. For *test results*, show a log of one or more sessions in which all user options are tested.

7. *Test results* must show that the code works correctly for all menu options and a wide variety of input data.

8. Project presentation should include a binder with separators and a table of contents, separating user manual, design documentation, program code, and test results.

9. You may choose to add features to the above specification. Be sure to document these.

10. Note that the semester project is only part of the programming that students are expected to do as lab work for CS I. It may be important for your work on the project that you solve ungraded programming problems (see Homeworks 1-7) in order to learn the skills needed to meet the project specifications.

11. The project is designed to be done in stages. It is not acceptable to skip these stages and submit only the entire project at the end of the semester.

12. As you work, you will discover bugs and fix them. Save a log of your bugs and how you found and fixed them; you will use this log in Homework 7.

In a business setting, the application described here would probably be implemented using a database management system. This programming project is an exercise in which you will develop and practice your programming skills and explore some ways that databases may be implemented.

Requirements for submissions

The project will be submitted in stages throughout the semester as homework assignments.

- Each submission must include a description of the *process* by which you did the work, such as what difficulties you had, what new skills you learned, what new concepts became clearer, what work if any is not completed, etc. One part of process documentation can be *muddy concepts* feedback, in which you describe what is least clear in the classroom presentation and textbook, as relates to the project work.

- Submit process documentation, listings of program code, *and* test results, in a *thin* binder at end of semester.
- Code, documentation, and input/output should all be clear and readable; presentation counts.
- Comments should express *clearly* what the program does. Document program's name and purpose at top in a comment; document any function definitions similarly. Place your name, the date, and the assignment and problem numbers at the top of each source file.
- Use a standard indenting scheme.
- Homework *must* be submitted on time for credit. "It is *strongly* recommended that you start very early on each homework assignment and *ask questions* if any part is unclear or seems difficult." (syllabus)
- You are responsible for submitting only your own work and keeping your own work to yourself so that others can work independently. Work that contains copied code or comments will be returned ungraded.

Grading criteria for preliminary and final submissions

- Process documentation
- Specification documentation
- Design documentation, including descriptions of classes, modular breakdown of problem, and module hierarchy chart
- Correctness and completeness of code (user input/output, file I/O, branches, loops, subprograms, use of structure types)
- Test results
- Formatting and presentation of code.

Recipe for one way to organize the semester project

There are many correct ways to do this project. Below is one plan.

1. Restate the project specs in your own words and make them a comment at the start of your program file.
2. Read steps 3-5 below and summarize them in a design documentation comment.
3. Declare a web-page structure type or class, with members as described in the project spec.
4. Declare a web-page collection structure type or class, with an array of web-page items and an integer (number of web pages in the collection) as members.
5. Create a menu that lets user choose to input one new web page, display the collection, save the collection to file, or read a collection from file.
6. Test your menu and functions.
7. Add searching and sorting features as able to do so based on classroom discussion and handouts.

Homework 0 (basic CS I review)

Due 9/6

1. Write a C/C++ program with inputs x and y , that displays a figure, as below, where there are a total of y stars and each star is x spaces to the right of the one above it.
3. Log in at <http://framingham.blackboard.com>, "enroll" in 63.252 Computer Science II, and respond to message posted by instructor on the Discussion board for this course.

Examples:

$x=1, y=3$:

*

 *

 *

$x=3, y=4$:

*

 *

 *

 *

2. Write a program that prompts for integers, until the user enters 0, and *after all inputs* displays the largest input value and the average.

Documentation	/ 20
Blackboard reply	/ 20
#2 correctness	/ 20
#3 correctness	/ 20
Testing	/ 20

Homework 1 (Types, control structures, subprograms)

Due 9/16

Graded:

1. Begin the semester project (see Project handout) by writing first drafts of the User Manual and the design documentation. The User Manual should begin with a summary of what the program will do.

If you have not written a menu-driven, modular program before, your design may be sketchy. Include a module hierarchy diagram based on the idea that each user option in the main menu (except "Quit") will be a module.

Write code for the main loop, including menu display, user input, and the *switch* statement that follows. In the *switch* statement you will call functions corresponding to the menu choices. Declare and define these functions as stubs that do nothing more than display the fact that they have been called.

Ungraded:

2. Write a short C or C++ program that displays a table of the first 20 powers of 2 ($2^1, 2^2, \dots, 2^{20}$), formatted below using *bitwise operators* (not the *pow* function) to calculate the entries in the right-hand column of the table. The first three rows will be:

n	2 to the nth power
1	2
2	4
3	8

3. Write a program that prompts for a person's first name, up to 80 characters, and display it backwards.
4. Write a function that takes a string parameter and swaps its first and second words, using loops to iterate. Define words as sequences of characters separated by a space.

Text of main:

```
char s[] = "Jane Smith";  
swap_strings(s);  
cout << s << endl;
```

Output:

```
Smith Jane
```

5. Write a program with no input that for $n = 1$ to 100, displays $\frac{n^2(n+1)^2}{4}$ and $\sum_{k=1}^n k^3$ and the difference between them.

Output:

n	quotient	sum	diff
1	1	1	0
2
...			

Specification	/ 20
Design	/ 20
Menu	/ 20
Loop	/ 20
Stubs	/ 20

Homework 2 (Structure types and classes)

Due 9/24

Graded:

1. Define in your project source code a structure type or class to implement the abstract data type of a web-page object. Instead of “dates visited,” for this version use only one date (a string). In *main*, declare an instance of this type, and pass it as a parameter (possibly a reference parameter) to the functions that implement the menu choices of adding a web page and displaying a web page. Complete the definitions of these functions. Submit tests of the resulting program. Specs, process, and design documentation need not be resubmitted.
2. *Validate* the date; i.e.,
 - check to see if there are exactly two slashes;
 - see that some value from 1 to 12 is before the first slash;
 - ...1 to 31 between slashes;
 - 2-digit year is in third.

Ungraded:

3. (a) Declare a *structure type*, with appropriate members, that may be used to declare variables to represent customers. A customer has a customer ID, a name, address, and amount of last purchase. *Declare* associated global functions, with appropriate parameters, to input and display member data. (You need not *define* these functions.)
(b) Define the equivalent *class*, with member function declarations. (No function definitions needed.)

4. See the program *courseereg.cpp* in the CS II network directory and on the slide, “A course-registration class.” Add to it a declaration and definition of a Boolean member function, *is_null()*, that tells whether the *student_ID* or *course_ID* member of a registration object is 0, returning *true* (1), if so, otherwise returning *false* (0). Modify *main* to use a loop to input and display course-registration objects until the user enters a 0 and *is_null* thus returns *true*.

Sample I/O:

```
---Type 0 to exit---
Student ID, course ID: 124 63152
124 63152
Student ID, course ID: 329 63252
329 63252
Student ID, course ID: 782 43320
782 43320
Student ID, course ID: 0 0
```

Documentation	/ 10
Structure type	/ 30
Instance	/ 10
Functions	/ 40
Testing	/ 10

Homework 3 (Arrays)

Due 10/6

Graded:

1. In your semester project, implement a *collection* of web-page records as follows and submit test results, documenting your changes. Note that modularity is *required*; your collection must pass between functions as a class member or parameter, *not* as a global variable.
 - Declare a structure type or class that has two members: an array of instances of your web-page structure, and an integer representing the number of elements of this array that are currently in use.
 - Declare an instance of this type in your *main* in place of the instance of the web-page type.
 - If you use a structure type, pass this *collection* object as a value or reference parameter to each of the functions called from *main*. If you use a class, the collection will be accessible to all member functions.
 - Define *two* of these functions properly: the one that inputs one record and adds it to the collection, and the one that displays all records.
 - *Suggestion*: in the functions that take the collection as a parameter, call the functions that display and input *one* web site object.

Ungraded:

2. Write a program that prompts for a series of integers, then stores them in an array, then
 - (a) displays the number of 13's found among the input values;
 - (b) tells whether the array is in ascending order.
3. Write a C or C++ program that uses a two-dimensional array, initialized as below, to calculate and display sales summaries by quarter and by region for a company with these sales, in millions of dollars.

	Q1	Q2	Q3	Q4
Atlantic	34	21	29	42
Midwest	82	78	92	74
West	157	183	256	173

A table like the following should be displayed. The *Year* column and the *All regions* row should be *calculated* by the program, not initialized by the programmer. It's easiest if loops are used.

(Sales in millions of dollars)

	Q1	Q2	Q3	Q4	Year
Atlantic	34	21	29	42	126
Midwest	82	78	92	74	326
West	157	183	256	173	769
All regions	273	282	377	289	

4. Write and test a subprogram that accepts as parameters an unsorted array of integers, and the size of this collection, and returns the *second* largest value in the array.

Documentation	/ 15
Collection structure type	/ 25
Input	/ 25
Display	/ 20
Testing	/ 15

Homework 4 (File I/O)

Due 10/21

Graded:

1. Extend your project by writing the definitions of the functions that save and retrieve web-site records. Write and test functions to retrieve and save your array-based collection of web-page records for the project.
 - If you use a class, write member functions of the collection class such that function *save* opens a file for output and loops through your array, calling a function *save* of the web-page class once for each element of the collection.
 - The *save* for web page records should take a parameter that is the file stream object. When the collection *save* function has finished the file-writing loop, it should close the file.
 - Write a similar collection function, *retrieve*, that opens a file for input and loops until *eof()* to call a *retrieve* function of the web-page class.
 - *Document* by explaining data file format.
 - *Test* by inputting several records, saving and ending the session, then retrieve and display the results. Show a log of save, retrieve, and display operations (you need not show hard copy of input tests).

Ungraded:

2. Write a program to read a set of floating-point values in a file, as below, and display these values and their *maximum*. The program should read as many or as few values as are in the file.

Sample output:

```
29.3 42.9 56 19.8 62.1 9.0 49.4  
Average: 38.357
```

3. Write a program that reads a text file named by the user, line by line, and copies it to another text file, named by the user, which is a copy of the original input file except that each line begins with its line number followed by a colon.

Sample partial output, given user input of "echofil.cpp":

```
1: // echofile.cpp  
2: // Reads a text file and displays its  
  contents.  
3: #include <iostream.h>  
4: #include <fstream.h>  
5: void main()
```

Documentation	/ 20
Menu	/ 20
Retrieve	/ 20
Save	/ 20
Testing	/ 20

Homework 5 (Searching and sorting)

due 10/29

Graded:

1. Implement the menu option *Search* in your project, using a linear search. Allow the user to specify an ID or name and display each web-page that matches the search key.
2. Under the *Display* option, display a menu to allow the user to choose an attribute on which to sort the array; sort the array and display the collection as sorted on that attribute.
3. *Optional challenge:* permit search on date; display all web pages visited on a date input by the user. This will require an additional search operation that traverses your linked list of dates.
4. *Optional challenge:* Implement a *Sort* menu option that sorts the collection and sets a flag, *is_sorted*. The *is_sorted* flag should be cleared any time a new item is input or retrieved. Now write *search* as a binary search; but only execute this search if the *sorted* flag is set.

Ungraded:

5. Write a program that reads a series of floating-point values from a text file, *floats.txt*, into an array and displays them, sorted ascending.
6. Write a function that takes four integer arrays as parameters and merges the first three into the fourth.
7. Write a program that initializes or inputs an array of *char* and displays the length of the longest run of consecutive 1s. (*Challenge:* Use random data to fill your array.)
Example: with the input data, '0', '0', '1', '0', '1', '1', '1', '0', '1', '0', '0', the program will output "3", because the longest run of 1s is the series of the 5th, 6th, and 7th values above.

Documentation	/ 20
Linear search	/ 20
Sorts	/ 20
Display	/ 20
Testing	/ 20

Homework 6 (Pointers and dynamic allocation)

due 11/12

Graded:

1. Modify your project so that the web-page structure type has a *linked list* of dates, rather than a single date. You will need to modify all functions of the web-page structure type, but no functions of the collection type.

Suggestion: write a date-node structure type that has as members a string (the date) and a pointer to the next date node; write a group of functions that display a list of dates, save a list of dates to a file, and retrieve a list of dates from a file.

Ungraded:

2. Using the following type declaration for points on a coordinate graph:

```
struct Point
{
    int x,y;
};
```

write and test a function that prompts for the x and y members of a point object, dynamically allocates a point on the heap, copies the input to the dynamic variable, and returns a pointer to the dynamic variable. Your driver in *main* should display that point and should contain the following line:

```
point* p = get_point();
```

If user input is "4 5", output should be "(4, 5)".

3. Write and test a function, *capitalize*, that accepts a parameter that is a pointer to *char*. The function should convert the character to upper case so that if *main* passes to *capitalize* a pointer to a character variable with value 'q', the variable will come back as 'Q'.

4. (a) Change your implementation of the web-page project to replace your array of web pages with an array of *pointers* to web pages. When a web page is input by the user or retrieved from file, first dynamically allocate an instance of your web-page structure type. Whenever you access your array of pointers, to access a site's information you must *dereference* the pointer. If your array is declared

```
site A[1000];
```

then when you access the *price* member of the book object with subscript i , for example, you must code $A[i] \rightarrow price$.

- (b) Write another version that statically allocates an array of pointers to books and then assigns to each pointer the address of a dynamically allocated book object. You need only submit your *main* function or other function that performs the dynamic allocation.

Documentation	/ 20
Pointer declarations	/ 20
Dynamic allocation	/ 20
List	/ 20
Testing	/ 20

Homework 7 (Software development issues)

due 11/22

Graded:

1. Separate your project source code into at least two separate *.cpp* files and compile them separately, linking them together as a Visual C++ *project*. To do this, you will have to create one or more *.h* files for constant and structure-type definitions, and use *#include* directives to make the definitions available to all the *.cpp* files that use them.
2. In your process description for this assignment, describe as many bugs as you can recall from doing this project, and describe the process of finding and fixing them.

Ungraded:

3. Write pseudocode or C or C++ code for the push and pop operations on a stack of *char* implemented as a dynamically allocated C-style string.
4. Modify one of the example programs, *intstack.cpp*, *balance.c.*, *balance.cpp*, or *pointstk.cpp*, or write your own stack code, so that your program will prompt the user for a *push*, *pop*, or *quit* option. Your *push* should prompt for a value and store it on the stack; *pop* should pop and display the top value; *quit* should terminate the program. *Note*: You should not need to modify any function except *main* of any of the above programs.

5. Write a program that responds to three user options: 'e' (enqueue), 'd' (dequeue), or 'q' (quit). When the user inputs 'e', your program should prompt for an integer that the program will insert at the tail of a linked list. When the user inputs 'd', your program should display the character that is at the head of the list and remove it from the list. At the end of program execution, your program should display any character that are still in the list.

Sample I/O:

```
Option? e
Enter integer: 5
Option? e
Enter integer: 2
Option? e
Enter integer: 7
Option? d
5
Option? e
Enter integer: 3
Option? d
2
Option? d
7
Option? q
3
```

#1	/ 60
#2	/ 40

Homework 8 (Languages and language processing)

due 12/3

Choose one of the following, including documentation that relates your code to the language concepts presented in the lectures:

1. Implement a calculator by writing a *command interpreter* for the following language:

- *add n* (adds *n* to accumulator)
- *sub n* (subtracts *n* from accumulator)
- *mul n* (multiplies accumulator by *n*)
- *div n* (divides accumulator by *n*)
- *out* (displays contents of accumulator)

In the list above, *n* is any numeric literal.

Assume that the accumulator storage location starts with a 0 in it and can store any *double* value. Test your interpreter with a series of commands.

2. Write a program that will read a file containing LISP program code and will output a file that contains a list of the lexical tokens in the LISP file, together with the lexical category for each token. The lexical categories are as follows:

- left parenthesis;
- right parenthesis;
- apostrophe;
- numerals (series of digits);
- identifiers (series of letters).

For example, if the LISP file is

```
(defun quicksort(slist)
  (cond ((null slist) nil)
        (t (append
             (quicksort (keep (car slist)
                              (cdr slist)))
             (cons (car x) (quicksort
                        (keepgt(car slist)
                              (cdr slist))))
             )
        )
  )
)
(quicksort (5 4 1 8 3 9 0 2))
```

then the output file will begin and end as follows:

```
(      lparen
defun   ID
quicksort ID
(      lparen
slist  ID
)      rparen
...
3      num
9      num
0      num
2      num
)      rparen
```

To write your program, you will need to use a buffer string to store the current token you are building, character by character. For example, if you read a letter from the file, read more characters until you reach a non-letter, and save all those letters as an identifier. Numerals have the same form as identifiers, except they are composed of digits. If you read a parenthesis, that is a token by itself. Discard all white-space characters (space, tab, newline).

You will need to use an input expression, such as *fin.get()*, that gets precisely one character at a time, or else use a line-by-line input expression such as *fin.getline()*, and scan through the input line character by character.

Code documentation	/ 20
Documentation relating this code to concepts presented in lectures	/ 20
Design & code	/ 40
Testing	/ 20

Homework 9 (Computer science and discrete probability)

(Ungraded)

To be determined

[debug problems to disperse thruout course]

1. In a phrase or sentence, tell what is the syntax or logic error, or the instance of questionable programming style in each of the programs below.

```
// err01.c
#include <stdio.h>
```

```
struct Employees
{
    quantity;
    oyees emp;
};
```

```
void main(void)
{
    oyees e1,e2;
}
```

```
// err02.c
#include <stdio.h>
```

```
void display(int list[]);
```

```
void main(void)
{
```

```
list[] = {2,34,82,44,18,30};
```

```
y(list[2]);
}
```

```
void display(int list[])
{
```

```
("%d\n",list);
}
```

```
// err03.c
#include <stdio.h>
```

```
void display(int item);
```

```
void main(void)
{
```

```
Bob = 2;
```

```
y(Bob);
}
```

```
void display(int item)
// Displays <item>.
```

```
{
    ("%d\n",item);
}
```

```
// err04.c
#include <stdio.h>
```

```
void display(int item);
```

```
void main(void)
{
```

```
= 2;
```

```
y(q);
}
```

```
void display(int item)
// Displays <item>.
```

```
{
    ("%d\n",item);
}
```

```
// err05.c
#include <stdio.h>
```

```
int quantity;
```

```
void display();
```

```
void main(void)
{
```

```
("Quantity? ");
```

```
("%d",&quantity);
```

```
y();
}
```

```
void display()
// Displays <quantity>.
```

```
{
    ("%d\n",quantity);
}
```

int

Empl

empl

int

displa

printf

int

displa

printf

int q

displa

printf

printf

scanf

displa

printf

```
// err06.c
#include <stdio.h>

void display(int num);

void main(void)
{
    int
    quantity;

    printf
    ("Quantity? ");
    scanf
    ("%d",&quantity);
    displa
    y(int quantity);
}

void display(int num)
// Displays <num>.
{
    printf
    ("%d\n",num);
}

// err07.c
#include <stdio.h>

void show(int arr[],int size);

void main(void)
{
    int
    item[] = { 13,54,8,28,31};

    show
    (item);
}

void show(int arr[],int size)
// Displays <num>.
{
    int i;
    for
    (i=0; i < 5; ++i)

    printf
    ("%d\n",arr[i]);
}
```

```
// err08.c
#include <stdio.h>

void show_total(int arr[],int size);

void main(void)
{
    int
    item[] = { 13,54,8,28,31};

    show
    _total(item);
}

void show_total(int arr[],int size)
// Displays <num>.
{
    int
    i,total;

    for
    (i=0; i < 5; ++i)

    total
    += arr[i];

    printf
    ("%d\n",total);
}

// err09.c
#include <stdio.h>
void show(int arr[],int subscript);
void main(void)
{
    int
    item[] = { 13,54,8,28,31};

    show
    (item,index);
}

void show(int arr[],int subscript)
// Displays <subscript>th item
// of <arr>.
{
    printf
    ("%d\n",arr[subscript]);
}

// err10.c
#include <stdio.h>

void main(void)
{
```

```
name1[80],name2[80];
("Two names please: ");
("%s %s",name1,name2);
(name1 > name2)

char
printf      ("%s",name1);
scanf
if          ("%s",name2);
           (" is greater.\n");
           }
```

printf
else
printf
printf

(Random access;)

1. Write a design in pseudocode for an abstract data type to implement an inventory database that uses random file access to store and retrieve records. (See the parts-manager problem from earlier homeworks.) What structure types will be needed? What operations (functions) will be implemented? It should be possible to update a large inventory database in real time or to quickly search it on any field.
2. The program below sorts a random-access file of characters. Modify it to create and sort a random-access file of strings. For test purposes, you may wish to create a short sequential text file, read the strings from it and write them to a random-access file, and sort it.
3. (Optional challenge) Modify your parts manager from an earlier homework to use random file access for storage and retrieval of records.