ECE 309 Spring 2024

Program 1: List update problem

Due Date: February 9th 2024 (11:59PM)

1- **Assignment Learning Objectives:**

- Linked-list programming and manipulation.
- Mapping real problem into data structures.
- Solving problems in an object-oriented manner.

2- **Associated Files:**

- Assignment description (this document)
- Test input files (e.g., request.txt)
- Linked-list data file (data.txt)
- Template makefile (Makefile)
- Validation output results (validation.txt)

Both files will be on Moodle under (Programming 1 Assignment)

3- **Description:**

The List update problem focuses on the strategy to reorder the list so that it can minimize the total number of steps to access the values associated with different objects. For instance, as shown in Figure 1 below, the number of steps to access the item with value 6 is 5 accesses.

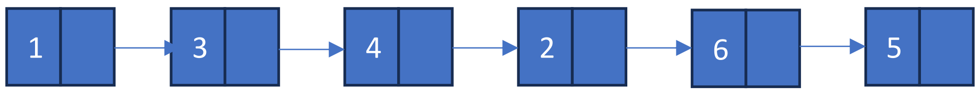


Figure 1 Example Linked-List Ready for Traversal

However, if we know that certain items (e.g., that with value 6) are accessed more frequently than others, then we would rather change the arrangement of the linked. For example, one arrangement is that the items are linked in a descending fashion, such that item with value 6 comes first, then that of value 5, etc. Accordingly, in this assignment we will explore ways to dynamically rearrange the linked-list order with the aspiration to reduce the number of steps of future accesses. To do so, you will need to implement two different methods:

- **Move-to-front approach:** in this approach, upon an access to an item, we want to assume that it is very likely it will be accessed again soon, and thus we move it to the front of the linked list, as shown in Figure 2 below:

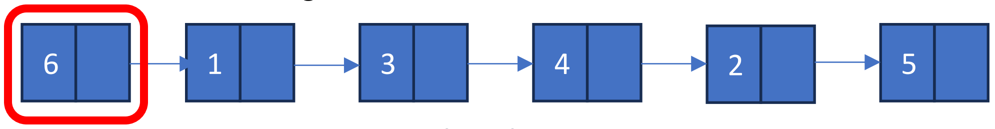


Figure 2 Example Move-to-front after accessing the item with vale 6.

- **Transpose:** in this approach, upon an access to an item, we want to assume that it is very likely it will be accessed again soon, however instead of putting it in the front we just advance its location by one position, i.e., we swap it with the one immediately preceding it (if any). In other words, we are conservatively moving it close to the front, but not immediately to the front position. Figure 3 below captures the Transpose approach:

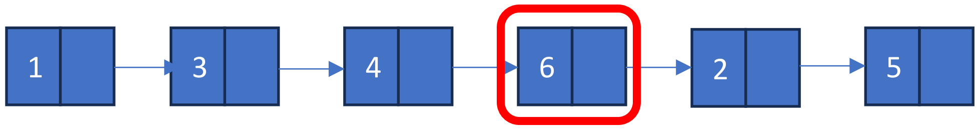


Figure 3 Transpose after accessing the item with value 6.

To test your implementation, your program should expect three input parameters, one with the repositioning heuristic (0: for Move-to-front, 1 for Transpose) and the second input is the name of the input file including the requests to access the linked list. The input file is provided as mentioned earlier (called request.txt). Finally, the 3rd parameter is the filename for the linked-list data itself, which your program will parse to create the initial linked list. Thus, to execute your program it takes the following form:

./program <repositioning approach> <input filename> <linked list file>

For example:

./program 0 request.txt data.txt

The program that you will implement shall do the following:

1- Parse the input parameters to know the heuristic to be used, input file, LinkedList file
2- For the above example, the program will parse the data.txt file to create the initial linked list. Each line in the file represents an integer value to be stored in a linked-list item.
3- Iterate over each item in request_x.txt to process the requests; each line is an independent request for a particular item in the linked list, to be identified through the item value.
4- After processing each request in (3), you will use the heuristic option (in this example Move-to-front) to decide how to update the linked-list based on the request. For this example, you will move the accessed item to the front.

5- Move to the next request
6- Repeat 5-6 until the end of the request_x.txt file

To understand the impact of a certain heuristic, your program should output a single value as an integer (followed by end line, i.e., cout<<output_value<<endl;), which represents the total number of steps taken for all accesses. You will be provided with multiple request input files, and in your report, you will compare the number of steps for each heuristic (i.e., move-to-front vs. transpose) and provide your rationale for the reason of difference in number of steps between heuristics.

Notice:
you can use validation.txt to check whether your code run correctly. For example, validation1.txt contains 10 requests from 0 to 9, which have 55 total number of steps for move-to-front and 55 total number of steps for transpose, and validation2.txt contains 8 requests (01302024), which have 23 total numbers of steps for move-to-front and 22 total numbers of steps for transpose.

4- <u>Report Format:</u>

The report should include:

- comparison of total number of steps for each request file

	request_1.txt	request_2.txt	request_3.txt
Move-to-front			
Transpose			

- your rationale for the reason of difference in number of steps between heuristics.

Please submit your report in PDF format with file name “report.pdf”.

5- <u>Submission Format:</u>

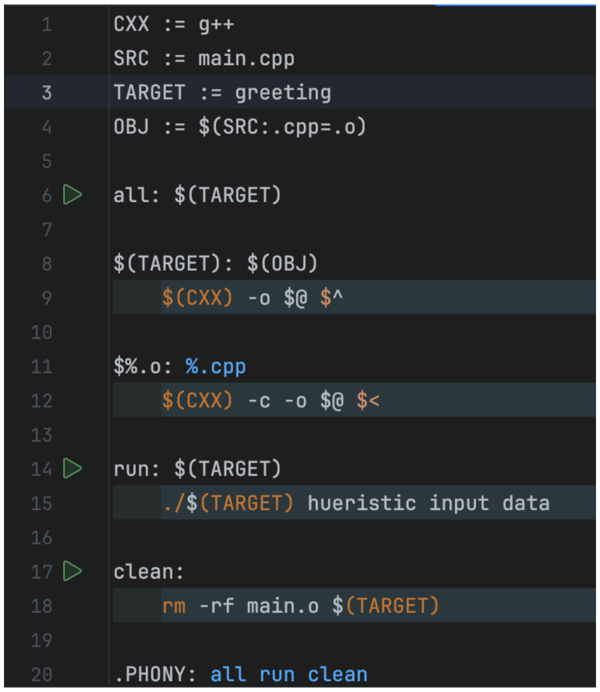
A folder “prog1” includes: main.cpp, Makefile, request_1.txt, request_2.txt, request_3.txt, and report.pdf.
Please make sure Makefile can work before you submitted. TA will use it to compile and run your code.

6- <u>How to use Makefile:</u>

Makefile template will help you compile your code and create an executable file for you so that you can check whether the program will run correctly. In the following, we will briefly introduce some basic commands to help you understand how to work with it.

1. Let’s say, we have a code that print out “Welcome to ECE309” in main.cpp.

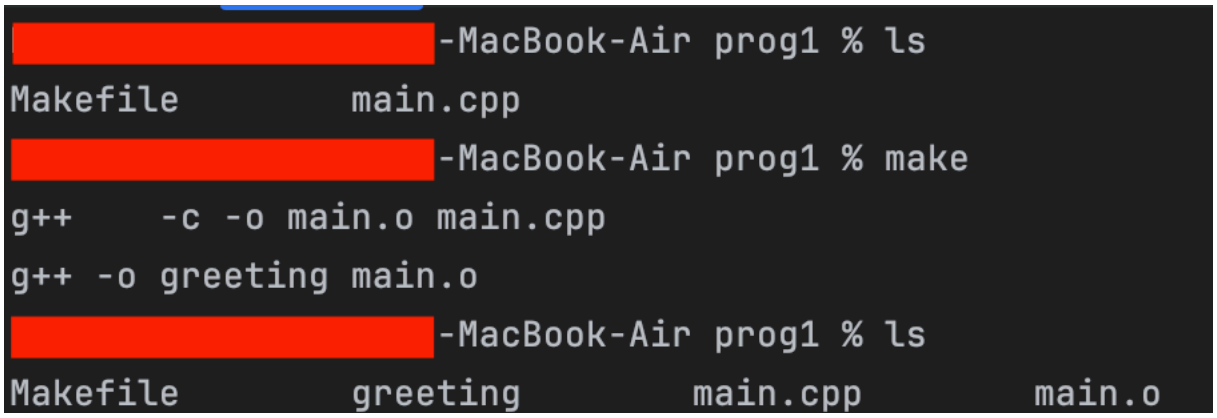
2. To create an exeuctable file name greeting, we specify the name to TARGET in the Makefile.



```
CXX := g++
SRC := main.cpp
TARGET := greeting
OBJ := $(SRC:.cpp=.o)

all: $(TARGET)

$(TARGET): $(OBJ)
	$(CXX) -o $@ $^

$%.o: %.cpp
	$(CXX) -c -o $@ $<

run: $(TARGET)
	./$(TARGET) hueristic input data

clean:
	rm -rf main.o $(TARGET)

.PHONY: all run clean
```

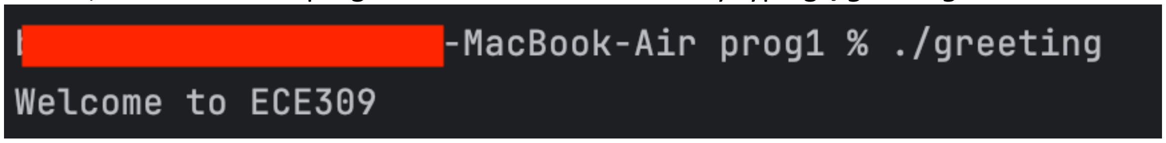
3. By typing ***make*** in the command line, which you can open a terminal from CLion in your project, the executable file "greeting" is created.

Notice that whenever we make changes in the code, we can type ***make*** again to create the executable file with the latest version.



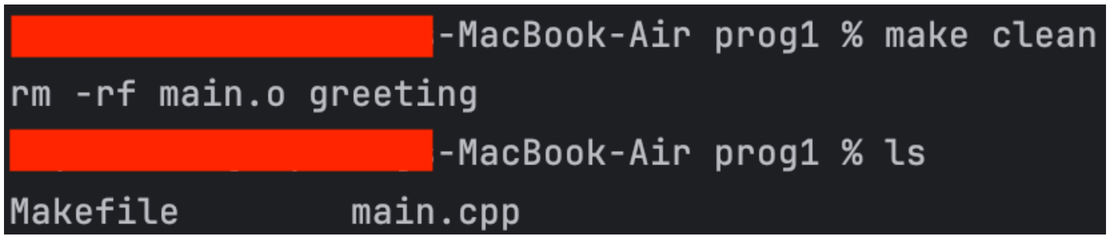
```
[redacted]-MacBook-Air prog1 % ls
Makefile        main.cpp
[redacted]-MacBook-Air prog1 % make
g++     -c -o main.o main.cpp
g++ -o greeting main.o
[redacted]-MacBook-Air prog1 % ls
Makefile        greeting        main.cpp        main.o
```

4. Then, we can run the program in the command line by typing ***./greeting***



```
[redacted]-MacBook-Air prog1 % ./greeting
Welcome to ECE309
```

5. To clean all the .o file and executable file that you just created, type ***make clean***



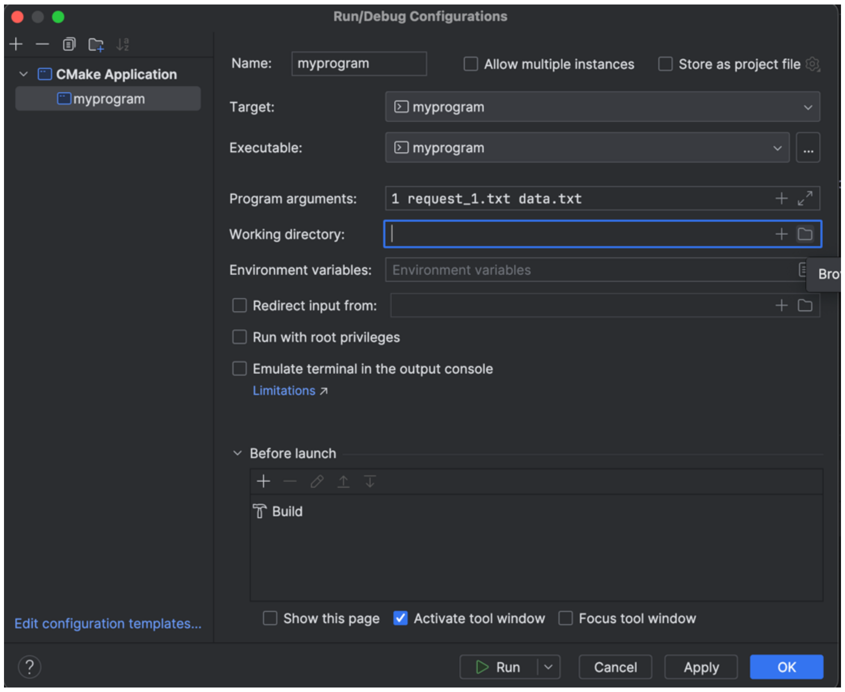
```
[redacted]-MacBook-Air prog1 % make clean
rm -rf main.o greeting
[redacted]-MacBook-Air prog1 % ls
Makefile          main.cpp
```

In this assignment, you can use the Makefile template that we provided to run in the command line. Basically, you just need to put it in the same directory as main.cpp and change the name in TARGET (for example, list_update instead of greeting). This will create an executable file named “list_update” and run the program by typing ***./list_update heuristic request_x.txt data.txt***

7- <u>Tips for developing program in CLion:</u>

1. For students who are not familiar with CLion, you can first create the project. (For example: myprogram)

2. Since our program will take three inputs, you can set up the inputs (also the working directory)whenever you run it in CLion so that you don’t have to use Makefile all the time.

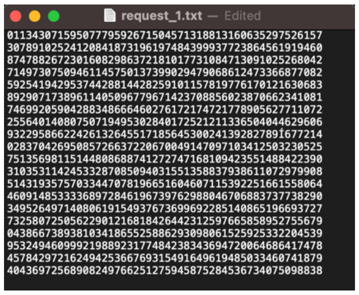


Name: myprogram
Target: myprogram
Executable: myprogram
Program arguments: 1 request_1.txt data.txt
Working directory:
Environment variables: Environment variables
☐ Redirect input from:
☐ Run with root privileges
☐ Emulate terminal in the output console
☐ Show this page ☑ Activate tool window ☐ Focus tool window

Go to “Run” menu and select “Edit Configuration.”

3. By setting this configuration, you can easily test your code by clicking RUN in the toolbar.

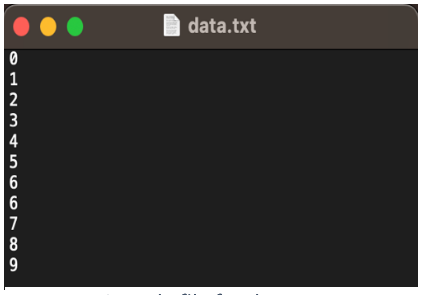
8- Sample file for input:



request_1.txt — Edited

```
01134307159507779592671504571318813160635297526157
30789102524120841873196197484399937723864561919460
87478826723016082986372181017731084713091025268042
71497307509461145750137399029479068612473366877082
59254194295374428814428259101157819776170121630683
89290717389611405096779671423708856023870662341081
74699205904288348666460276172174721778905627711072
25564014080750719495302840172521211336504044629606
93229586622426132645517185645300241392827891677214
02837042695085726637220670049147097103412503230525
75135698115144808688741272747168109423551488422390
31035311424533287085094031551358837938611072979908
51431935757033447078196651604607115392251661558064
46091485333368972846196739762988046706883737738290
34952649714080619154937673699692285140865196693727
73258072505622901216818426442312597665858952755679
04386673893810341865525886293098061525925332204539
95324946099921988923177484238343694720064686417478
45784297216249425366769315491649619485033460741879
40436972568908249766251275945875284536734075098838
```

Sample file for request_1.txt



data.txt

```
0
1
2
3
4
5
6
6
7
8
9
```

Sample file for data.txt

9- Grading:

10 points: Files submitted with proper name.
10 points: Program is complete. The program successfully takes three inputs and returns the integer as output.
10 points: Proper coding style and comments.
60 points: the program performs all functions correctly:
- (10 pts) Create a class named **Linkedlist** and store the data.txt in it.
- (10 pts) Correctly implement for Move-to-front.
- (10 pts) Correctly implement for transpose.
- (10 pts) Correct results for validation1.txt and validation2.txt for
- (20 pts) Correct results for request files.

10 points: Report.

Good luck