Project 3: Linked List

Corrections and Additions

- (7/14) Added sample output for non-existing filenames
- (7/20) Added test scripts

Files

1. generate.c
2. mylist_template.c

Learning Goals

The purpose of this assignment is to quickly become good at writing C programs, gaining the experience of working in a non-object oriented language. By the end of this assignment you should be comfortable with arrays, command-line arguments, file I/O, pointers, and structures in C. Additionally you will learn how to implement a linked list and some of its basic operations such as list traversal, node addition, and sorting a linked list.

Specifications

In this assignment, you will write a C program named `mylist.c` that reads a list of integers from one or more input files, stores them in a linked list, then builds a sorted linked list via an array and then writes the sorted output to a new file.

Generate integers and store it in a file

To begin with, you are given the program `generate.c`, which generates a random set of integers and prints out every integer in its own line to the console (stdout). The program takes two command line arguments:

- `<total_nums>` The number of integers to be generated
- `<max_num>` The upper bound on the highest value

To use this program to generate the input file for list.c do the following:

- Create a folder named p3 in your private/354 directory that you created for assignment 2.
- Copy the generate.c file into your p3 directory. You may find this file here: generate.c
- Compile the program and store the executable in a file named generate.
- Run the program using the command: `./generate 20 100`
- For the purpose of this program we will keep the number of integers (20 in this case) below 1000 which is defined as a symbolic constant `MAX_INTS` in mylist.c.
- The program should have printed out a list of integers to stdout, separated by |.
- Alter the shell command such that the output sent to stdout is instead captured in a file. This is called a shell redirect. `./generate 20 100 > numbers.txt`
- The program should have created a file named `numbers.txt`, which contains a list of integers written as ASCII characters.
- To try to see the results, you might try the shell command: `cat numbers.txt`

Here's a sample output:

```
$ gcc generate.c -o generate -m32 -Wall
$ ./generate 20 100 > numbers.txt
$ cat numbers.txt
| 66 | 40 | 81 | 41 | 12 | 58 | 21 | 40 | 35 | 43 | 74 | 43 | 17 | 4 | 96 | 62 | 92 | 48 | 98 | 59 |
```

Intro to mylist.c

For this and the next parts of the assignment you have been provided with the file mylist_template.c which contains skeleton code for the functions that you need to complete to make the program to work as expected. Copy the file into your p3 directory.

This program takes in multiple command line arguments `./mylist <-i|-d> <OUTPUT_FILE> <FILE>...`

- `<-i|-d>` The first argument specifies how the linked list should be sorted. -i stands for incrementally while -d stands for decrementally.
- `<OUTPUT_FILE>` The second argument specifies where your program should store the contents of your linked list.
- `<FILE>...` One or more input files are specified. These files are input files, which are the names of files containing lists of integers to sort.

Your program first needs to verify that the number of command line arguments is as expected. For example, options other than `-i` or `-d` is invalid; having no input file is not valid, either. If your program encounters invalid command line arguments, a usage message (provided by `const char *USAGE`) should be printed out.

Read integers from input files and build a linked list

The program takes multiple filenames that contain lists of integers to sort. Here you will:

- Open a file at a time whose name is given as a command line argument. The file is guaranteed to have the same format as the output of generate.c.
- Read the line of integers. You may want to use the function `getline`. Check out `man getline` for help.
- Tokenize the string into integers. You might want to use the function `strtok` or `strtok_r`. Similarly please check out the man pages via `man strtok`.
- Insert the integers you get from the file to the beginning of your linked list.
- Print the content of the linked list. You may use the function `list_to_string`.
- Do the same thing for every other input files. Insert the integers to the beginning of the list.

To accomplish this part, you will complete the following functions:

```
void insert(node_t *head, int data)
```

This function takes the dummy head of your linked list, appends a new node after the dummy head with value `data`.

```
ssize_t build_list_from_file(node_t *head, const char *filename)
```

This function should read the file with name `filename`, tokenize the line read from the file, and insert the integers using the insert function. It returns the number of integers inserted to your list upon success. If the file cannot be opened or does not exist, your function should return -1 and print the error message written in `const char *BADFILE`.

Convert the list to an array

Next you will convert the list to an array (for efficient sorting later). You will complete the following function:

```
void list_to_array(int *integers, node_t *head)
```

This functions takes the head node and adds all integers to the array pointed by integers. You should allocate a large enough array so that errors such as buffer overflow won't occur. We restrict a file to only contain at most `MAX_INTS` (1000) integers. Think about how this may help you allocate the array.

Sort the integers

The time complexity of sorting a linked list is extremely high compared to that done in an array. So, you will be required to sort the array of integers. Note that the order (whether incrementally or decrementally) is determined by the command line flag -i or -d. Fortunately, C offers a quick sort library function to help you out. Check out `man qsort` for more information on how to use this function. We provide two helper functions `qsort_inc` and `qsort_dec`. You may use them as two comparator functions for incremental and decremental sorting.

Create a new sorted list

Now you should convert the sorted array to a new list. You may use the following function

```
void array_to_list(node_t *head, int *integers, size_t len)
```

Here `head` is a pointer to the new head node, `integers` is the integer array, and `len` is the length of the integer array. Think about how to use the output from one of the previous functions to determine the value of `len`.

Write the output to a file

Now you will implement the following function to write output to a file.

```
int write_to_file(const char *filename, node_t *orighead, node_t *newhead)
```

Here `filename` is the filename of the output file. Feel free to truncate it to zero length if the file already exists. `orighead` is the original unsorted list, and `newhead` is the sorted list. This function returns 0 on success and -1 on failure. Errors occurred during `fopen` should be considered as failure. We have provided the function `list_to_string` that returns a string with integers in a formatted form. Please check out the sample output section below about the format of the output file.

Sample output

We provide some sample output here

```
$ gcc generate.c -o generate -m32 -Wall
$ ./generate 2 10 > 2-10.txt
$ ./generate 3 50 > 3-50.txt
$ ./generate 10 500 > 10-500.txt
$ gcc mylist.c -o mylist -m32 -Wall
$ ./mylist -i
Usage: ./mylist <-i|-d> <OUTPUT_FILE> <FILE>...
$ ./mylist -i output.txt 10-500.txt
LIST CONTENT AS OF FILE 10-500.txt: head->|243|->|35|->|440|->|21|->|258|->|12|->|241|->|381|->|240|->|166|->NULL
$ cat output.txt
ORIGINAL LIST: head->|243|->|35|->|440|->|21|->|258|->|12|->|241|->|381|->|240|->|166|->NULL
SORTED LIST: head->|12|->|21|->|35|->|166|->|240|->|241|->|243|->|258|->|381|->|440|->NULL
$ ./mylist -d output.txt 2-10.txt 3-50.txt 10-500.txt
LIST CONTENT AS OF FILE 2-10.txt: head->|0|->|6|->NULL
LIST CONTENT AS OF FILE 3-50.txt: head->|31|->|40|->|16|->|0|->|6|->NULL
LIST CONTENT AS OF FILE 10-500.txt: head->|243|->|35|->|440|->|21|->|258|->|12|->|241|->|381|->|240|->|166|->|31|->|40|->|16|->|0|->|6|->NULL
$ cat output.txt
ORIGINAL LIST: head->|243|->|35|->|440|->|21|->|258|->|12|->|241|->|381|->|240|->|166|->|31|->|40|->|16|->|0|->|6|->NULL
SORTED LIST: head->|440|->|381|->|258|->|243|->|241|->|240|->|166|->|40|->|35|->|31|->|21|->|16|->|12|->|6|->|0|->NULL
$ ./mylist -i output.txt 10-500.txt badfile123 2-10.txt
LIST CONTENT AS OF FILE 10-500.txt: head->|243|->|35|->|440|->|21|->|258|->|12|->|241|->|381|->|240|->|166|->NULL
ERROR: unable to process file badfile123
LIST CONTENT AS OF FILE 2-10.txt: head->|0|->|6|->|243|->|35|->|440|->|21|->|258|->|12|->|241|->|381|->|240|->|166|->NULL
```

Tests

We provide some tests under the folder P3 Linked List/test. Please download the entire folder and place it inside the same directory along with your c file. Your directory tree should look like

```
├── mylist.c
└── test
    ├── 100-1000.txt
    ├── 10-500.txt
    ├── 2-10.txt
    ├── 3-50.txt
    ├── out2.test
    ├── out3.test
    ├── out4.test
    ├── out5.test
    ├── p3.py
    ├── stdout1.test
    ├── stdout2.test
    ├── stdout3.test
    ├── stdout4.test
    └── stdout5.test
```

To test your code, type in `python3 test/p3.py`. If you would like to continue the test suite if a test failed, then type in `python3 test/p3.py -c`.

Requirements

- Your program should operate exactly as the sample output shown above.
- Use a CSL Linux machine for this assignment!
- We will compile your program with gcc -Wall -m32 on a CSL Linux machine. So, your program must compile there, and without warnings and errors. It is your responsibility to ensure that your program compile on the department Linux machines, and points will be deducted for any warnings or errors.
- Remember to think broadly and do error handling correct for your program. Read the spec carefully and think about where errors might occur in your program.
- Start working ASAP!
- Style_Guide.pdf