## Comprehensive experiment 1：Application of linear list

## One. Experimental target

1、Use the basic operations to implement the specific operations for the linear table;

2、Master the application of file operations;

3、Improve the understanding of the data structure of linked storage structure, and gradually cultivate the programming ability to solve practical problems.

## Two. Experimental environment

A computer with Visual C ++ 6.0 / CFree.

This experiment has 4 class hours in all.

## Three. Experiment content

(Choose one from the following three contents)

### address list design

Design a classmate's address list, requested as follows:

* Each student in the address list contains the following information: student id、name、telephone number. If you need more fields, please add them yourself.
* The program has a main menu containing the following functions:

1. Add a record: Add a student record from the input.
2. Delete a record: Delete a student record according to the student id from the input.
3. Output all records: Display all the records in the address list.
4. Search by name: Input the student name and then output the whole information of the student.
5. Save records: Save all the records in the address list to a certain file.
6. Clear records: Delete all the records in the address list and then delete the file.
7. Quit

**hint：**

* When the program starts, it should be determined whether there is a record file. If the file exists, read each record from it to the list.
* After the user selects and completes a function of the main menu, the program should return to the main menu.
* When a record is added, it should be inserted into the tail of the list.
* If a record does not exist when performing delete or and search operation, the program should output some information to the user.
* You do not need to write files when adding records or deleting records.
* When you want to save a record you’d better overwrite the file. (Or delete the original file first, and then save all the records)
* Each module is written in the form of a function, called by the main function.

**optional：**

* Add a sorting function in the main menu, the sorting result should be in an ascending order according to the student number. Sorting methods can be done by bubble sort or insert sort.

### Expression calculation

Design an automatic calculator, which is described as follows:

（1）The expressions that need to be calculated are stored in a TXT file;

（2）Each line in the text is an expression;

（3）Expressions include operands, addition, subtraction, multiplication, division, and parentheses;

for example:（34-72.3）\*54.7-82.4

（4）The "automatic calculator" calculates each expression in the text file according to the input file name, and stores each result to the file named [original filename]\_new.txt. You’d better overwrite the file when saving the records. Each row in the file should be written in the following format:

For example: the original file is A1.txt

The calculated output file is: A1\_out.txt

The result in A1\_out.txt text is in the form of：

（34-72.3）\*54.7-82.4 = -2177.41

If the calculation result is decimal, it should hold to 4 decimal places.

（5）When the calculation is completed, a statistical file should be generated. Its content is as follows：

Execution time: xxxx-xx-xx hh:mm:ss

The total number of expressions is: XXX

The number of correct expression is: XXX

The number of incorrect expressions is：XXX

The rule of naming the statistical file: [original filename] \_log.txt, and the file is written in additional write mode.

For example: If the original filename is A1.txt file, then the statistical document should be: A1\_log.txt

### Design of a scheduling queue

There is a Scheduling System and its detail is shown below:

1. Scheduling tasks are saved in a text file and each line has only one task.
2. Each scheduling task contains the task name, the task start time, and the total time required to perform the task.
3. Assume that each task can get a 2-minute execution time. When the execution time is up, it is moved to the end of the scheduling queue, waiting for a new round of execution assignments.
4. If a task is finished, it is written into the log file: task\_log.txt. The format is:

The task name The task start time The task end time The scheduling frequency

**Tips：**

√ Open the text file where the task is located, store the tasks in chronological order to the schedule queue;

√ Take the tasks out from the scheduling queue one by one, then modify the frequency and subtract the execution time by 2 minutes. If greater than 0, then append the task to the tail of queue after the modification of its execution time.

√ If (the execution time is) less than or equal to 0, the task is completed and the results of its execution are written to the task\_log.txt file.

## Four. Requirement

1、Submit the experimental reports in groups (no more than 3 persons in each group).

2、Submit the source code individually for submission. The file name is named as:

Long student ID\_Name\_CE1.doc OR Long student ID\_Name\_CE1.pdf

The report template is shown as follows:

**XXX Experiment Report**

Class: network2

Student ID 1: 18401190125 Name : 徐炜涛 Experiment Date:2019-9-28

**Experimental purpose**

1. **Experimental environment**

1、Use the basic operations to implement the specific operations for the linear table;

2、Master the application of file operations;

3、Improve the understanding of the data structure of linked storage structure, and gradually cultivate the programming ability to solve practical problems.

1. **Experimental content**

A computer with Visual Studio

This experiment has 4 class hours in all.

(See the instruction manual for the above three parts)

1. **Important data structures**

Design a classmate's address list, requested as follows:

* Each student in the address list contains the following information: student id、name、telephone number. If you need more fields, please add them yourself.
* The program has a main menu containing the following functions:

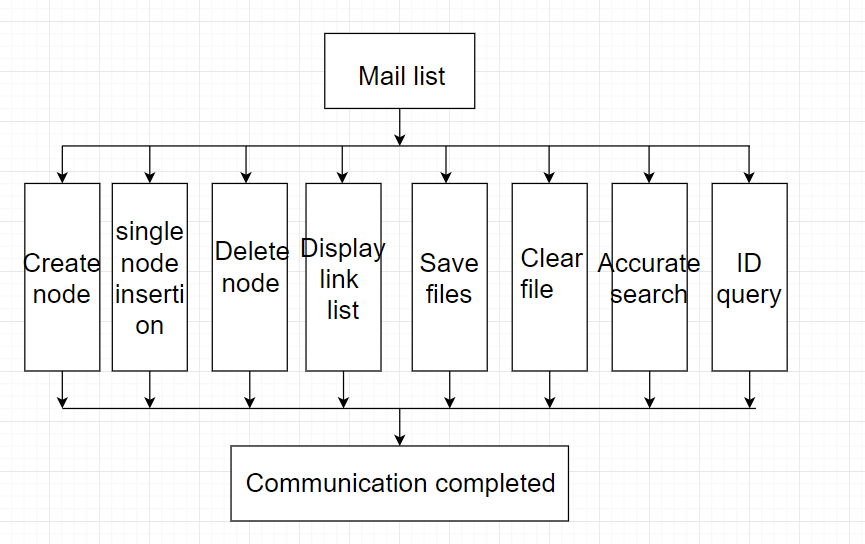
1. Add a record: Add a student record from the input.
2. Delete a record: Delete a student record according to the student id from the input.
3. Output all records: Display all the records in the address list.
4. Search by name: Input the student name and then output the whole information of the student.
5. Save records: Save all the records in the address list to a certain file.
6. Clear records: Delete all the records in the address list and then delete the file.
7. Quit

**hint：**

* When the program starts, it should be determined whether there is a record file. If the file exists, read each record from it to the list.
* After the user selects and completes a function of the main menu, the program should return to the main menu.
* When a record is added, it should be inserted into the tail of the list.

1. **Implementation analysis**

|  |
| --- |
| typedef struct stu {  int id; //id storage student ID  char name[20]; //storage student name  int tel; //storage student telephone  stu \*next; //Link to the next node  }stu; |
| stu\*Create(); // Bulk creation of data nodes  stu\*Insert\_(stu\*head, stu\*p); // Insert data nodes  stu\*Delete\_(stu\*head, int id); // Delete data nodes  stu\*Display\_(stu\*head); // Show all nodes  stu\*Find\_(stu\*head, char name[]); // Find the required nodes by name  stu\*Rough(stu\*head, char name[]); // Rough search, keyword search stu\*Store\_(stu\*head); // Store in a file  stu\*Clear\_(stu\*head); // Clear Nodes and Delete Files |
| stu\*Create();  stu\*Insert\_(stu\*head, stu\*p);// Because creating nodes is also an insertion  function  while (cin >> id && id != 0 && cin >> name >> tel ) // When you input 0 it will break this function  {  p = (stu\*)malloc(size);  p->id = id;  strcpy(p->name, name);  p->tel = tel;  head = Insert\_(head, p); //the main operation  } |
| stu\*Insert\_(stu\*head, stu\*p);  stu \*ptr, \*ptr1, \*ptr2;// Create two secondary pointers and one pointer to the accepted data  if (head == NULL) //Judging whether the list is empty  {  head = ptr;  head->next = NULL;  }  Else  while (ptr->id > ptr2->id&&(ptr2->next != NULL)) // Traverse to find smaller locations than incoming data  {  ptr1 = ptr2; // Recording Locations Finded Using Auxiliary Pointer  ptr2 = ptr2->next;  }  if (ptr->id <= ptr2->id)  {  if (head == ptr2) // Find whether the node found is in the header node head = ptr;  else  ptr1->next = ptr; // Not Head Node  ptr1->next = ptr2;  }  else  {  ptr2->next = ptr; // In the smallest case, insert the tail  ptr->next = NULL;  }  } |
| struct stu\*Delete\_(struct stu\*head, int id)  stu \*ptr1, \*ptr2; // Two auxiliary pointers, one for deletion  while (head != NULL && head->id == id) // If the first one is the data you are looking for  {  ptr2 = head;  head = head->next;  free(ptr2);  }  ptr1 = head;  ptr2 = head->next; // Delete the list before and after  while (ptr2->next != NULL)  {  if (ptr2->id == id)  {  ptr1->next = ptr2->next;  free(ptr2); //  }  else  ptr1->next = ptr2;  ptr2 = ptr1->next;  } |
| stu\*Clear\_(stu\*head)  fstream clear("学生信息.txt", ios::trunc); // Rewrite  if(remove("学生信息.txt") == 0) // Delete files  while (head) // Release all nodes  {  p = head;  p = head->next;  free(p);  head = p;  }  clear.close(); |
| stu\*Rough(stu\*head, char name[])  int count = 0; // According to keyword search, if there are keywords found, the data about keywords will be output.  for (p = head; p != NULL; p = p->next)  {  if (p->name[0]==name[0]||p->name[1]==name[1]||p->name[2]==name[2])  {  count = 1;  }  } |
| In order to prevent duplication of storage after reading, there will be operations to clear the contents of files after reading the contents of files to the linked list.  while(check >> id >> name >> tel)// If there is information in the text read out and stored in the linked list  fstream clear("学生信息.txt", ios::trunc); |



|  |
| --- |
| data structure：  A student has many kinds of information, so it is necessary to create a data structure to store the various information of his classmates. A structure is used in the topic and stored in chain mode. |
| Main function：  The project requires the creation of addressbook to add deletion, modification and save files, clear files and other functions, and return to the page after the operation. From the conditions, we can draw the conclusion that the various conditions here are juxtaposition relations. Here we use switch statement to distribute different functions to different values. |
| Functions:  The insertion function is the most important among all the completed functions. The insertion function I designed can be sorted automatically according to the size of id, and then linked to the linked list. It is completed by using two auxiliary pointers and one acceptance data pointer. The program that creates the linked list only calls the insertion function on the basis of batch creation to preserve it. The order after certificate creation, other search and deletion functions are to traverse the entire list to display (or delete) the data of information such as ID or name or keyword. When the file reads the list, in order to prevent it. |

1. **Debugging problem analysis**

**1：**

Problem：

When debugging for the first time, there was no return value of a head set over the insert side, resulting in the data displayed each time after insertion.

Solvetion：

Define each function as the type of the structure, and finally return the header pointer so that each operation is traversed from scratch.

2：

Problem：

When checking whether the input telephone number is legitimate, a judgment statement is used. If the inconsistency is not satisfied, the goto statement is directly used to return to the place where the input was started and re-enter, but the test has a dead cycle.

Solvetion：

Because the goto statement is more likely to happen unexpectedly, so cancel this function, increase the input loop, use count = 0, when the phone number is qualified, jump out of the dead cycle, if not successful, prompt for re-entry, until successful, the following operation is not carried out

3:

Problem：

After reading the file data and putting it in the linked list, the error of inserting duplicate information occurs during insertion, which results in two or more identical information in the text.

Solvetion：

Write a function that checks for errors. Each insert calls a query to see if all information is the same. It does not insert, it does not continue to operate.

1. **Summary**

This experiment knowledge summary and own experience

1. **Crew Division**

|  |  |  |
| --- | --- | --- |
| **Group division** | | |
| **Member name** | **Work done** | **Completion situation** |
| **徐炜涛** | **Code Design, Implementation, Document Writing, Final Summary** |  |
|  |  |  |
|  |  |  |