Large unsigned integers processor

B11002208
彭聖堯

Contents

Problem, Definition, and Details

- **Read an integer file to build the linked lists.**
  - Read file part:
    - Use fopen_s,fclose,fseek,fgets to get file's data until EOF.
  - Build linked list part:
    - Use char array to Implement big number store.
    - Use strtok_s to split file's data and create node.
    - Always append nodes at the end of the linked list.
- **Sum all the distinct integers.**
  - An simple big number add algorithm.
- **Count the number of integers.**
  - Traverse from root node until NULL.

- **Remove duplicate integers for each list and print out the resulting lists.**
  - An worst efficiency(O(n^2)) way to deduplicate:
    - Just a comparison between each other.
- **Find the largest integer for all integers in the lists.**
  - Use a Independent node to store largest number.
  - Use multiple way to compare number’s:
    - 0.sign(Not implemented)
    - 1.length
    - 2.strcmp
- **Print out the list.**
  - Simply to implementation with recursion.

Code

```
#include "node.h"

int main()
{
	char mode;
	while (1)
	{
		printf("%s",prompt);
		scanf_s(" %c", &mode, 1);			// get user enter as mode select
		while ((getchar()) != '\n');
		clearOutput();				// clear the console output to make it look easier
		if (seleteOperation(mode))
		{
			return 0;
		}
	}
	return 0;
}
```

```
void myStrrev(char* str) //reverse string
{
	char* temp = str;
	while (*(temp++))
		;
	temp -= 2;
	while (str < temp)
	{//swap
		*str ^= *temp;
		*temp ^= *str;
		*str ^= *temp;
		str++;
		temp--;
	}
}
```

```
char* getStringFromFile(FILE* file) // allocate a char array to load file's string
{
	uint32_t size = 0;
	fseek(file, 0, SEEK_END);
	size = ftell(file);//get length
	fseek(file, 0, SEEK_SET);
	char* temp = calloc((uint64_t)size + 2, sizeof(char));
	if (temp)
	{
		fgets(temp, size + 2, file);
	}
	else
	{
		fprintf(stderr, "error at line %d,memory allocate failed", __LINE__);
		abort();
	}

	return temp;
}
```

```
int seleteOperation(char mode)
{
	switch (mode)
	{
		node* temp;
		case 'a':
			printf("Please enter a file name to load or only press enter to load default file.\ndefault is \"./number.txt\" (Max 50 Character):");
			fflush(stdin);
			*fileName = getchar();
			if (*fileName == '\n')// check is user custom or default
			{
				strcpy_s(fileName, 13, "./number.txt");
			}
			else
			{
				scanf_s(" %[^\n]", (fileName + 1), 50);
			}
			if (fopen_s(&dataSource, fileName, "r"))
			{
				printf("Load failed.\n");
				return 0;
			}   // open file
			printf("Loading...\n");
			char* string = getStringFromFile(dataSource); // try to load file
			root = loadNumFromString(root, NULL, string); // pharse string to linked list
			fclose(dataSource);                  // close file if file is opened
			free(string);
			printf("Load done.\n");
			return 0;
```

```
case 'c':
                temp = getLargestNode(root);              // get largest node from root
                if (temp)
                {
                    printf("largest node is:\n");
                }
                printNode(getLargestNode(root));          // use function to print node
                return 0;
            case 'd':
                root = deduplicate(root);                 // use function to deduplicate linked list
                printf("Deduplication done.\n");
                return 0;
            case 'e':
                sumAndPrintList(root);                    // a simple implementation of a big number adder
                return 0;
            case 'f':
                printf("Now list's data:\n");
                printf("==============================\n");
                printAllList(root);                       // print all node in this linked list
                return 0;
            case 'q':
                return -1;
            default:
                printf("unknown mode\n");
                return 0;
        }
        printf("================================================================================
======\n");
        return 0;
}
```

```
node* loadNumFromString(node* root, node* last, char* str)  //use token to split string and transform string to node
{
    static char* token = " ,";
    static char* tempPtr = NULL;
    static char* strtokBuffer = NULL;
    node* current = NULL;
    if (!root) //create root node if root is not exist
    {
        root = calloc(1, sizeof(node));
        if (root != NULL)
        {
            root->next = NULL;
            root->length = 0;
            root->sign = 1;
            root->numString = NULL;
            current = root;
        }
        else
        {
            fprintf(stderr, "error at line %d,memory allocate failed", __LINE__);
            abort();
        }
    }
    else if (!last) // init last node when first run
    {
        last = root;
        while (last->next)
        {
            last = last->next;
        }
    }
```

```
tempPtr = strtok_s(str, token, &strtokBuffer);  //take apart of string
    if (tempPtr)
    {
        if (!current)
        {
            current = calloc(1, sizeof(node));
            if (current)
            {
                current->next = NULL;
                current->length = 0;
                current->sign = 1;
                current->numString = NULL;
                last->next = current;
            }
            else
            {
                fprintf(stderr, "error at line %d,memory allocate failed", __LINE__);
                abort();
            }
        }
        if (*tempPtr == '-'){
            current -> sign = 0;
            tempPtr++;
        }
        current->numString = calloc(strlen(tempPtr) + 1, sizeof(char));
        if (!current->numString)
        {
            exit(EXIT_FAILURE);
        }
        strcpy_s(current->numString, strlen(tempPtr) + 1, tempPtr);  //copy string into node
        current->length = strlen(current->numString);
        return loadNumFromString(root, current, 0);
    }
    return root;
}
```

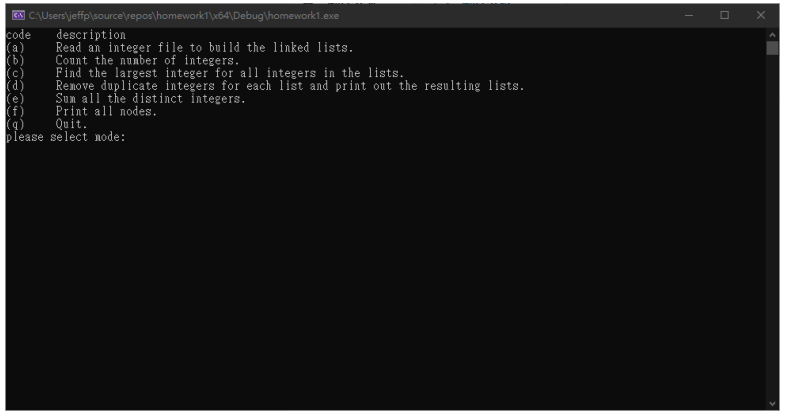
```
void printAllList(node* current) // a recursion function to print linked list
{
	if (!current)
	{
		printf("linked list is not exist!\n");
		return;
	}
	printf("%c%s\n",current-> sign ? 0:'-', current->numString);
	if (current->next)
	{
		printAllList(current->next);
	}
}

void printNode(node* current) //a normal function to print a single node
{
	if (!current)
	{
		printf("linked list is not exist!\n");
		return;
	}
	printf("%c%s\n",current-> sign ? 0:'-', current->numString);
}

node* getLargestNode(node* current) // use multiple conditions to find the largest node
{
	if (!current)
	{
		return NULL;
	}
	node* largest = current;
	current = current->next;
	for (; current; current = current->next)
	{
		if (largest -> sign > current -> sign){
			continue;
		}
		if (largest->length > current->length)
		{
			continue;
		}
		if (largest->length < current->length)
		{
			largest = current;
			continue;
		}
		int cmp = strcmp(largest->numString, current->numString);
		if ((largest -> sign && cmp < 0)||(!largest -> sign && cmp > 0))
		{
			largest = current;
			continue;
		}
	}
	return largest;
}
```

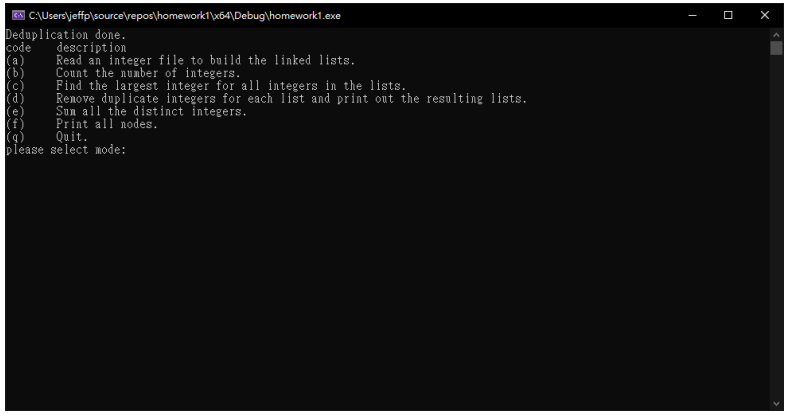
```
int nodeCount(node* root) // run until root(current) is NULL(false)
{
	int count = 0;
	while (root)
	{
		root = root->next;
		count++;
	}
	return count;
}

void addToFirstNode(char* sum, char* target) // add target to sum
{
	short sumEnd = 0, targetEnd = 0, carry = 0;
	while (!(sumEnd & targetEnd)) // main add part
	{
		*sum -= '0';
		*sum += carry;
		if (!targetEnd)
		{
			*sum += *target - '0';
			target++;
			if (!*target)
			{
				targetEnd = 1;
			}
		}
		carry = *sum / 10;
		*sum = *sum % 10;
		*sum += '0';
		sum++;
		if (!*sum)
		{
			if (!targetEnd || carry)
			{
				*sum = '0';
				*(sum + 1) = 0;
			}
			else
			{
				sumEnd = 1;
			}
		}
	}
}
```

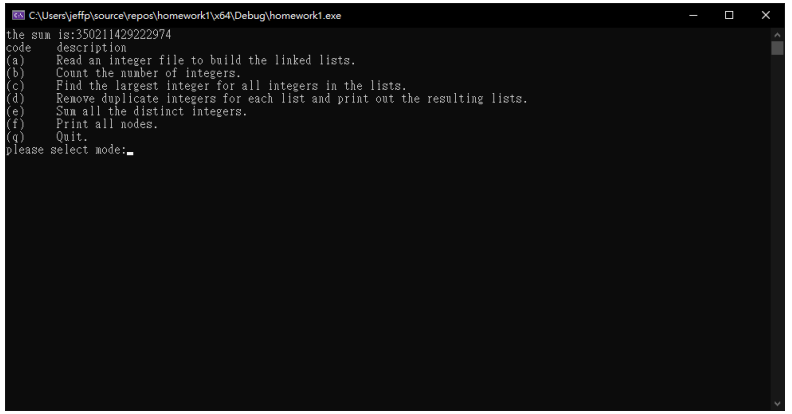
```
node* deduplicate(node* current) // an O(n^2) deduplicate funcction
{
    node* root = current;
    if (!root)
    {
        printf("linked list is not exist!\n");
        return NULL;
    }
    for (; current; current = current->next)
    {
        for (node* lastNode = current, *thisNode = current->next; thisNode; lastNode = thisNode, thisNode = thisNode->next)
        {
            if (current->length != thisNode->length) //length deff
            {
                continue;
            }
            if (strcmp(current->numString, thisNode->numString)) //context deff
            {
                continue;
            }
            lastNode->next = thisNode->next; // is duplicate
            free(thisNode);
            thisNode = lastNode;
        }
    }
    return root;
}
```

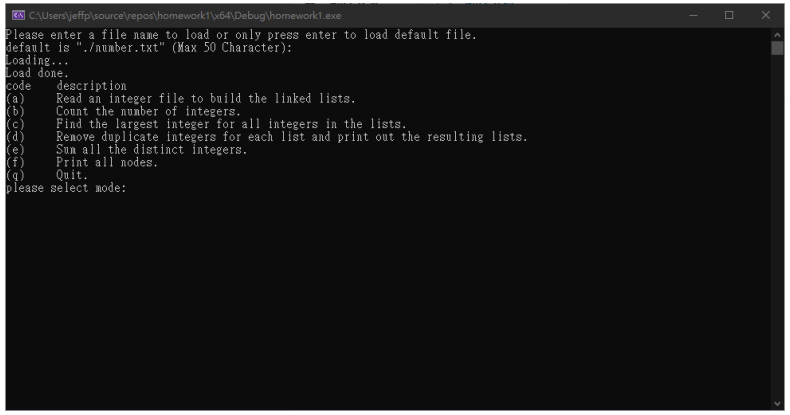
```
void sumAndPrintList(node* root)
{
    if (!root)
    {
        printf("linked list is not exist!\n");
        return;
    }
    node sum;
    sum.length = 1;
    sum.numString = calloc(2, sizeof(char));
    sum.numString[0] = '0';
    sum.sign = 1;
    sum.next = NULL;
    myStrrev(sum.numString);
    for (; root; root = root->next) //add each node
    {
        if (sum.length <= root->length) //extend node size if need
        {
            char* temp = realloc(sum.numString, sizeof(char) * (root->length + 2));
            if (temp)
            {
                sum.numString = temp;
            }
            sum.length = root->length + 1;
        }
        myStrrev(root->numString); //reverse string to add
        addToFirstNode(sum.numString, root->numString);
        myStrrev(root->numString); //reverse again to restore data
    }
    myStrrev(sum.numString);
    printf("the sum is:");
    printNode(&sum);
}
```

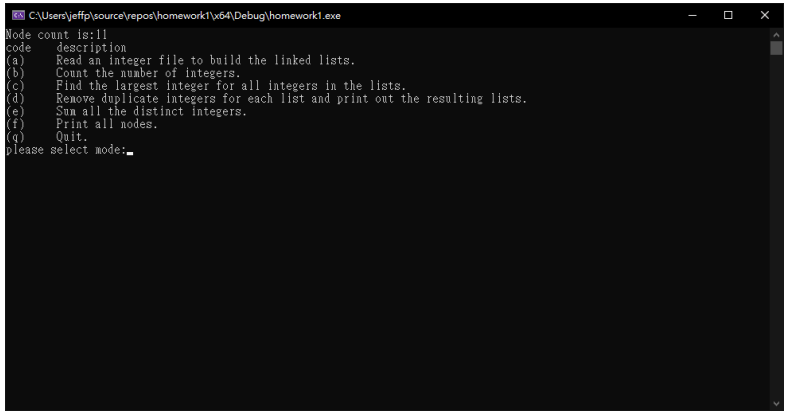
Results

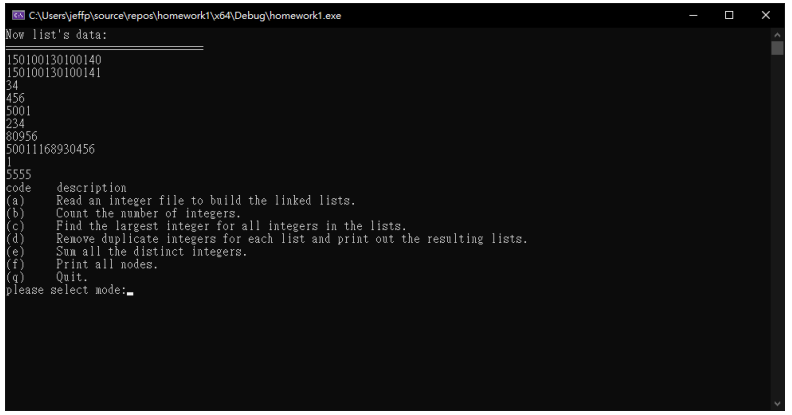

C:\Users\jeffp\source\repos\homework1\x64\Debug\homework1.exe
code description
(a) Read an integer file to build the linked lists.
(b) Count the number of integers.
(c) Find the largest integer for all integers in the lists.
(d) Remove duplicate integers for each list and print out the resulting lists.
(e) Sum all the distinct integers.
(f) Print all nodes.
(q) Quit.
please select mode:


C:\Users\jeffp\source\repos\homework1\x64\Debug\homework1.exe
Deduplication done.
code description
(a) Read an integer file to build the linked lists.
(b) Count the number of integers.
(c) Find the largest integer for all integers in the lists.
(d) Remove duplicate integers for each list and print out the resulting lists.
(e) Sum all the distinct integers.
(f) Print all nodes.
(q) Quit.
please select mode:


C:\Users\jeffp\source\repos\homework1\x64\Debug\homework1.exe
the sum is:350211429222974
code description
(a) Read an integer file to build the linked lists.
(b) Count the number of integers.
(c) Find the largest integer for all integers in the lists.
(d) Remove duplicate integers for each list and print out the resulting lists.
(e) Sum all the distinct integers.
(f) Print all nodes.
(q) Quit.
please select mode:


C:\Users\jeffp\source\repos\homework1\x64\Debug\homework1.exe
Please enter a file name to load or only press enter to load default file.
default is "./number.txt" (Max 50 Character):
Loading...
Load done.
code description
(a) Read an integer file to build the linked lists.
(b) Count the number of integers.
(c) Find the largest integer for all integers in the lists.
(d) Remove duplicate integers for each list and print out the resulting lists.
(e) Sum all the distinct integers.
(f) Print all nodes.
(q) Quit.
please select mode:


C:\Users\jeffp\source\repos\homework1\x64\Debug\homework1.exe
Node count is:11
code description
(a) Read an integer file to build the linked lists.
(b) Count the number of integers.
(c) Find the largest integer for all integers in the lists.
(d) Remove duplicate integers for each list and print out the resulting lists.
(e) Sum all the distinct integers.
(f) Print all nodes.
(q) Quit.
please select mode:


C:\Users\jeffp\source\repos\homework1\x64\Debug\homework1.exe
Now list's data:
150100130100140
150100130100141
34
456
5001
234
80956
50011168930456
1
5555
code description
(a) Read an integer file to build the linked lists.
(b) Count the number of integers.
(c) Find the largest integer for all integers in the lists.
(d) Remove duplicate integers for each list and print out the resulting lists.
(e) Sum all the distinct integers.
(f) Print all nodes.
(q) Quit.
please select mode:

Discussion and Conclusion

- I think I have many function's time/space complex is too high, like:
  - Deduplicate
    - Maybe can use hash map to improve performance.
  - loadNumFromString
    - If I can use int array to replace char array, is well save many memory and can calculate more faster.
- Big number is a classic topic about data structure, and I learn some trick when I doing this homework:
  - Use reverse's string can more easy to align
  - Pre-allocate node is better then allocate when need if memory is plentiful.