УНИВЕРСИТЕТ ИТМО
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Отчёт по лабораторной работе №4

Выполнил Ощепков Артём,
группа P3202
Преподаватель Жирков И.О.

Санкт-Петербург 2017

Текст задания

1.

The program accepts an arbitrary number of integers through stdin. What you have to do is

1. Save them all in a linked list in reverse order.
2. Write a function to compute the sum of elements in a linked list.
3. Use this function to compute the sum of elements in the saved list.
4. Free the memory allocated for the linked list.

You can be sure that

- The input does not contain anything but integers separated by whitespaces.
- All input numbers can be contained into int variables.

Following is the recommended list of functions to implement:

- list_create – accepts a number, returns a pointer to the new linked list node.
- list_add_front – accepts a number and a pointer to a pointer to the linked list.
- list_add_back, adds an element to the end of the list.
- list_get gets an element by index, or returns 0 if the index is outside the list bounds.
- list_free frees the memory allocated to all elements of list.
- list_length accepts a list and computes its length.
- list_node_at accepts a list and an index, returns a pointer to struct list,

corresponding to the node at this index. If the index is too big, returns NULL.

- list_sum accepts a list, returns the sum of elements.

2.

Implement several higher-order functions on linked lists:
foreach, map, map_mut, and foldl.
• foreach accepts a pointer to the list start and a function (which returns void and accepts an int). It launches the function on each element of the list.
• map accepts a function f and a list. It returns a new list containing the results of the f applied to all elements of the source list. The source list is not affected.
For example, f (x) = x + 1 will map the list (1, 2, 3) into (2, 3, 4).
• map_mut does the same but changes the source list.
• foldl is a bit more complicated. It accepts:
– – The accumulator starting value.
– – A function f (x, a).
– – A list of elements.
It returns a value of the same type as the accumulator, computed in the following way:
We launch f on accumulator and the first element of the list. The result is the new accumulator value a′.
We launch f on a′ and the second element in list. The result is again the new accumulator value a″.
We repeat the process until the list is consumed. In the end the final accumulator value is the final result.
For example, let’s take f (x, a) = x * a. By launching foldl with the accumulator value 1 and this function
we will compute the product of all elements in the list.
• iterate accepts the initial value s, list length n, and function f. It then generates a list of length n as follows: [s, f(s), f(f(s))]

3.

3.Implement foreach; using it, output the initial list to stdout twice: the first time, separate elements with spaces, the second time output each element on the new line.
4.Implement map; using it, output the squares and the cubes of the numbers from list.
5. Implement foldl; using it, output the sum and the minimal and maximal element in the list.
6. Implement map_mut; using it, output the modules of the input numbers.
7. Implement iterate; using it, create and output the list of the ten powers of two.
8. Implement a function bool save(struct list* lst, const char* filename);, which will write all elements of the list into a text file filename. It should return true in case the write is successful, false otherwise.
9.Implement a function bool load(struct list** lst, const char* filename);, which will read all integers from a text file filename and write the saved list into *lst. It should return true in case the write is successful, false otherwise.
10. Save the list into a text file and load it back using the two functions above. Verify that the save and load are correct.
11. Implement a function bool serialize(struct list* lst, const char* filename);, which will write all elements of the list into a binary file filename. It should return true in case the write is successful, false otherwise.
12.Implement a function bool deserialize(struct list** lst, const char* filename);, which will read all integers from a binary file filename and write the saved list into *lst. It should return true in case the write is successful, false otherwise.

Текст разработанной программы

```
list_node* list_create (cont list_content value){
    list_node* node = malloc(sizeof(*node));
    if(node) {
        node->value = value;
        node->next  = NULL;
    }
    return node;
}
void list_free( list_node**   list){
    list_node* to_free = (list_node*)*list;
    list_node* next;
    while(to_free){
        next = to_free->next;
        free(to_free);
        to_free = next;
    }
```

```
    (*list) = NULL;
}
// Index is changing. Is it normal?
list_node* list_node_at(size_t index, const list_node* const list){
    list_node* current_node = (list_node*)list;
    while(index--) {
        if(!current_node)
            return NULL;
        current_node = current_node->next;
    }
    return current_node;
}
list_content list_foldl (const list_node* const list,
                const list_content accumulator,
                list_content(*function)(list_content,list_content)){
  // Лучше ли использовать отдельную переменную аккумулятора или использовать то, что уже есть на стеке?
  // Функция принимает параметр, и поэтому не может изменить передаваемое значение. Так что явно приводим
  // Just for fun
    return list ?
                list->next ?
                   list_foldl(list->next, function((list_content)accumulator,  list->value),function )
                        :                   function((list_content)accumulator, list->value)
           : 0;
}
bool list_save_abstract(  list_node* lst, const char* const filename, const char* const file_mode,
                                                                    void(print_function)(list_content)){
    err = false;
    file = fopen(filename,file_mode);
    if(!file)
        return false;
    list_for_each(lst,print_function);
    fclose(file);
    return !err;
}
```

```
bool list_save(list_node* lst, const char* const filename){
   return list_save_abstract(lst,filename,"w",print_to_file);
}
```

Выводы:

Разрабатывая данную программу, я наткнулся на странный баг:

Предположим, имеется строка 10, в которой вызывается функция, падающая с seg fault'ом.

Перед ней, на строке 9, написано printf(«Hello»);

Ещё ранее, строкой 8, в stdout выводится содержимое некоторой структуры данных.

Что происходит при запуске данной программы?

В консоль ничего не выводится, за исключением сообщения о segfault'e.

Заменим printf(«Hello») на printf(«Hello\n»). Теперь после запуска мы видим в консоле

<содержимое_структуры> Hello

Ошибка сегментирования

Напрашивающиеся выводы:

printf осуществляет печать (замечу сразу, что сообщение появлялось и если вместо printf использовалось puts) в stdout. Причём, судя по результатам тестирования, вывод осуществляется сразу, если передан символ \n и позже, если его нет. В голову приходят два варианта объяснения (seg-fault-то на лишь следующей строке! (проверен ассемблерный код, компилятор не делает оптимизирующих перестановок)):

1) printf создаёт отдельный поток/процесс, но этот вариант отбрасывается по понятным причинам.

2) Где-то внутри записи в поток в ядрах linux'a имеется буфер, который наполняется и выкидывается ОС в сам поток либо при записи какого-либо специального символа, например, \n, либо по какому-либо прерыванию по решению ОС (не будет же printf ждать дополнения в виде конца строки, если оно встретится через часы выполнения кода). В принципе, вывод сходится с картиной мира и наличием функции а-ля flush в стандартной библиотеки.

Найденный баг не является багом, как таковым, а лишь тонкостью.

Зато появился замечательный вопрос на защиту лабы.