Санкт-Петербургский государственный политехнический университет
Кафедра “Прикладная математика”

Отчет по лабораторной лаботе 1

“Алгоритмы и сруктуры данных”

Студент группы № 5030102/20002

ФИО Двас Павел Григорьевич

Выполнил (дата) 15.10.2023

Оглавление

Постановка задачи 2

Описание алгоритма 2

Текст программы 2

Описание тестирования 6

Постановка задачи

Считать из текстового файла все строчки и заполнить список, содержащий следующую информацию: текст строки и его длина. Список должен заполняться таким образом, чтобы длины строк располагались в порядке возрастания. Вывести на экран 10 самых длинных строк. Ввести с клавиатуры число и проверить, есть ли в списке строка (или строки), имеющие такую длину.

Вывести их на экран.

Описание алгоритма

Для описания алгоритма требуется пояснить как работает односвязный список.

Односвязный список – структура данных, в которой каждый элемент (узел) хранит информацию, а также ссылку на следующий элемент. Последний элемент списка ссылается на NULL.

Для нас односвязный список полезен тем, что:

1. Он очень просто устроен и все алгоритмы интуитивно понятны
2. Односвязный список – хорошее упражнение для работы с указателями
3. Его очень просто визуализировать, это позволяет "в картинках" объяснить алгоритм

Односвязный список состоит из узлов. Каждый узел содержит значение и указатель на следующий узел, поэтому представим его в качестве структуры

```
typedef struct tagLIST LIST;

struct tagLIST
{
  char *Str;
  int StrLen;
  LIST *Next;
};
```

Где char *Str – содержимое строки, int StrLen – длина строки, LIST *Next – указатель на следующий элемент списка

Добавление элемента

Для добавления нового узла необходимо:

1. Выделить под него память
2. Задать ему значение
3. Сделать так, чтобы он ссылался на предыдущий элемент (или на NULL, если его не было)
4. Перекинуть указатель head на новый узел.

Поиск элемента

Для поиска элемента нужно:

1. Сделать цикл по всему списку
2. Сравнить элемент списка с искомым элементом
3. Если этот элемент не равен, переходим к шагу переписываем структуру на структуру со следующим указателем. Иначе, записываем нужные нам данные и процедура завершается.
4. Если элемент не был найден передаем в переменные пустые значения.

Печать односвязанного списка

Печать списка реализуется простым циклом

1. Передаем в функцию указатель на структуру.
2. Печатаем нужные нам данные вызовом метода в структуре

Текст программы

list.h:

```
/* Dvas Pavel, 5030102/20002 */

typedef struct tagLIST LIST;

struct tagLIST
{
  char *Str;
  int StrLen;
  LIST *Next;
};
void DisplayTenMost( LIST *L );

void DisplaySpecLen( LIST *L, int N );

int AddToListSorted( LIST **L, char *NewStr, int NewStrLen );
void DelFromListFront( LIST **L );
```

```
void ClearList( LIST **L );
```

list.c:

```
/* Dvas Pavel, 5030102/20002 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "list.h"

int ListLength;

void DisplayTenMost( LIST *L )
{
  int N = ListLength;

  if (L == NULL)
    printf("<empty list>\n");
  else
    while (L != NULL)
    {
      while (N > 10)
        L = L->Next, N--;
      printf("%s %i\n", L->Str, L->StrLen);
      printf((L->Next == NULL) ? "\n" : "");
      L = L->Next;
    }
}

void DisplaySpecLen( LIST *L, int N )
{
  if (L == NULL)
    printf("<empty list>\n");
  else
    while (L != NULL)
    {
      if (L->StrLen == N)
      {
        printf("%s %i\n", L->Str, L->StrLen);
        printf((L->Next == NULL) ? "\n" : "");
      }
      L = L->Next;
    }
}

int AddToListSorted( LIST **L, char *NewStr, int NewStrLen )
{
  LIST *NewElement;

  if ((NewElement = malloc(sizeof(LIST))) == NULL)
    return 0;
  if ((NewElement->Str = (char *)malloc(NewStrLen)) == NULL)
    return 0;

  if (*L == NULL)
  {
    strcpy(NewElement->Str, NewStr);
    NewElement->StrLen = NewStrLen;
    NewElement->Next = *L;
    *L = NewElement;
    ListLength++;
```

```
    return 1;
  }

  while ((*L)->Next != NULL && (*L)->StrLen < NewStrLen)
    L = &(*L)->Next;

  if ((*L)->StrLen < NewStrLen)
    L = &(*L)->Next;

  strcpy(NewElement->Str, NewStr);
  NewElement->StrLen = NewStrLen;
  NewElement->Next = *L;
  *L = NewElement;
  ListLength++;
  return 1;
}

void DelFromListFront( LIST **L )
{
  LIST *Old = *L;

  if (*L == NULL)
    return;
  *L = (*L)->Next;
  free(Old->Str);
  ListLength--;
  Old->StrLen = 0;
  free(Old);
  Old = NULL;
}

void ClearList( LIST **L )
{
  if (*L == NULL)
    return;

  while (*L != NULL)
    DelFromListFront(L);
}
```

main.c:

```
/* Dvas Pavel, 5030102/20002 */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

#include "list.h"

void main( void )
{
  FILE *F;
  char ch;
  char *str;
  int str_len, cap = 10, i;
  LIST *L = NULL;

  if ((F = fopen("test.txt", "r")) == NULL)
    return;

  do
```

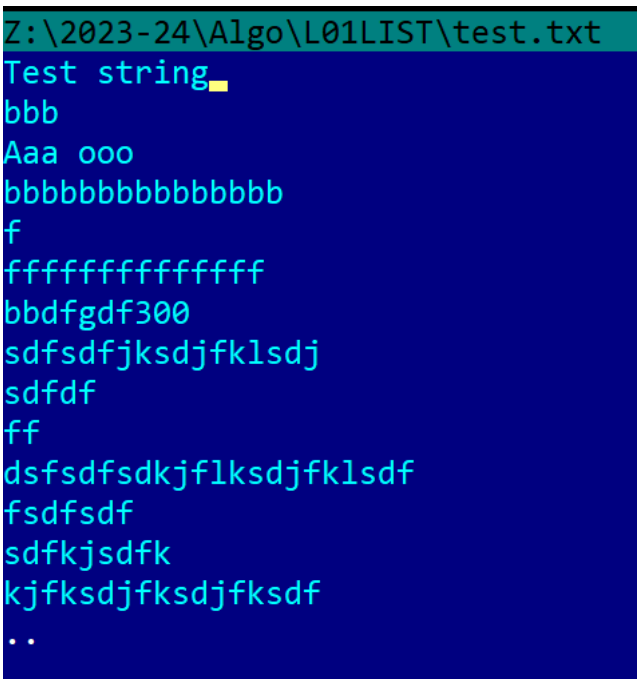
```
  {
    ch = fgetc(F);
    str_len = 0;
    if ((str = (char *)malloc(cap)) == NULL)
      return;
    while (ch != '\n' && ch != EOF)
    {
      str[str_len++] = ch;

      if (str_len >= cap)
      {
        cap *= 2;
        str = (char *)realloc(str, cap);
      }
      ch = fgetc(F);
    }
    str[str_len++] = '\0';
    AddToListSorted(&L, str, str_len);
    free(str);
  } while (ch != EOF);

  DisplayTenMost(L);

  printf("Check if string with specific length exists: ");
  scanf("%i", &i);
  DisplaySpecLen(L, i);
  fclose(F);
  ClearList(&L);
  _getch();
}
```

Описание тестирования

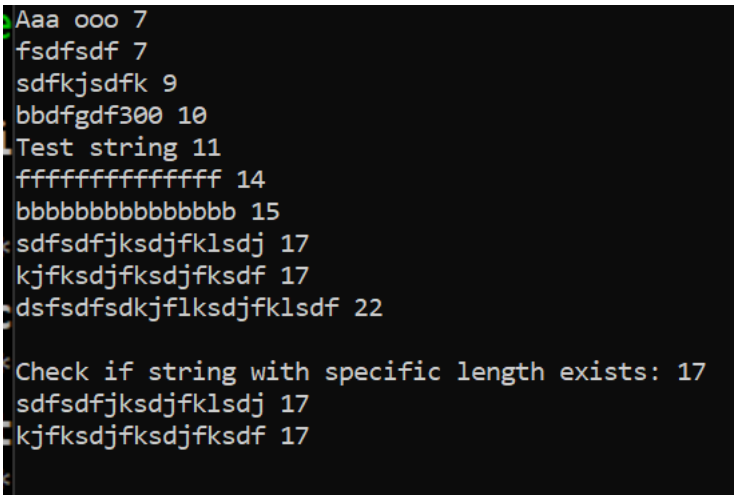
Для тестирования данной программы производился ее неоднократный запуск с вводом конкретных данных, при котором проверялось:

1) Стабильность работы программы при одинаковых входных данных;
2) Отсутствие “падений” и “зависаний”;
3) Корректное выполнение всех заявленных процедур;
4) Корректное завершение программы;

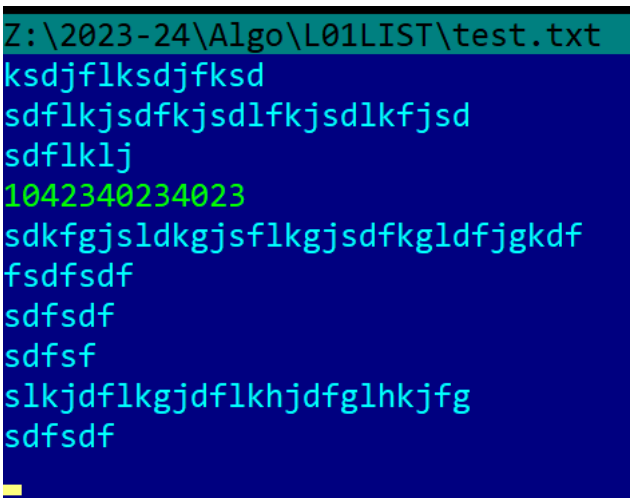
Приведем пример тестирования на скриншотах, данных ниже



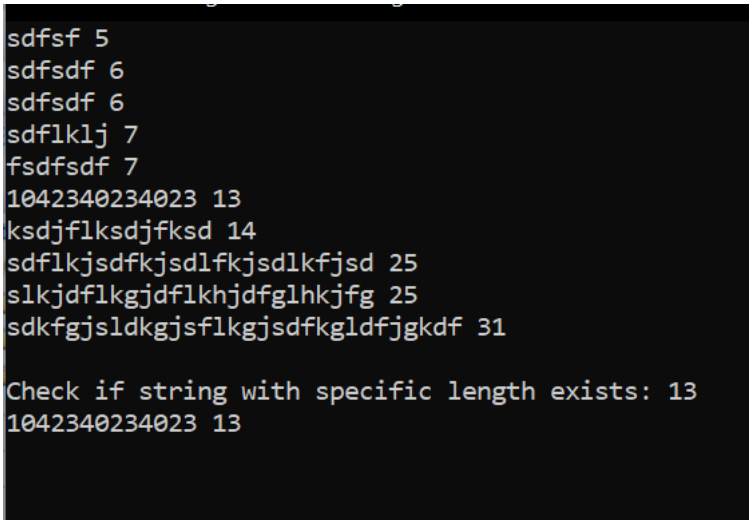
Исходный текстовый файл для теста



Пример работы программы



Исходный текстовый файл для теста



Пример работы программы

При проведении тестирования такого рода никаких проблем обнаружено не было, что позволяет судить о корректности работы программы в целом.