Федеральное государственное бюджетное образовательное учреждение
высшего образования «Сибирский государственный университет телекоммуникаций
и информатики»

Факультет ИВТ

Кафедра вычислительных систем

Курсовая работа

«Разработка инструментов командной строки в ОС GNU/Linux»
Вариант 2.2 «Калькулятор командное строки»

Выполнил:
студент гр. ИС-242
Денисов М. А.

Проверил:
Старший преподаватель Кафедры ВС
Фульман В.О.

Новосибирск, 2023

Тема курсовой работы
Тема выбранного раздела.

Задание на курсовую работу
Разработать программу-калькулятор cmdcalc, предназначенную для вычисления простейших арифметических выражений с учетом приоритета операций и с расстановки скобок. На вход команды cmdcalc через аргументы командной строки поступает символьная строка, содержащая арифметические выражения. Требуется проверить корректность входного выражения (правильность расстановки операндов, операций и скобок) и, если выражение корректно, вычислить его значение. Арифметическое выражение записывает следующим образом: **A p B** или (**A p B**), где A – левый операнд, B – правый операнд, p – арифметическая операция. A и B – представляют собой арифметические выражения или вещественные числа, p = +|-|*|/. Например: ((((1.1 − 2) + 3) ∗ (2.5 ∗ 2)) + 10), (1.1 – 2) + 3 ∗ (2.5 ∗ 2) + 10. Пример вызова команды cmdcalc (обратите внимание, что входное выражение необходимо взять в кавычки!):

```
$ cmdcalc "3 * 2 – 1 + 3"
```

Полученный ответ может отображаться на экране, а также сохраняться в файле.

Критерии оценки

- **Оценка «удовлетворительно»:** не предусмотрены скобки и приоритеты операций.
- **Оценка «хорошо»:** учитываются приоритеты операций.
- **Оценка «отлично»:** учитываются приоритеты операций, допускаются скобки.

Указание к выполнению задания
Одним из подходов к решению указанной задачи может быть использование структуры данных «стек». Данная структура представляет собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришел — первым вышел»). Одним из примеров стека может служить детская пирамидка: ось, на которую одеваются кольца. Первым с такой пирамидки снимается кольцо, размещенное на ней последним. Размещение элемента в стеке обозначают операцией push, а извлечение элемента из стека – операцией pop. Для решения указанной задачи потребуется два стека, первый (num) будет использоваться для хранения операндов, тип элемента – вещественное число, второй (ops) – для хранения операций и скобок, тип элемента – символ. Рассмотрим алгоритм на примере выражения $\mathbf{1}.\,\mathbf{1} - (\mathbf{5} + \mathbf{2} * (\mathbf{2} + \mathbf{3}))/\mathbf{10}$. Пошаговый разбор алгоритма представлен на рисунке 1.

1. $\mathbf{1}.\mathbf{1} - (5 + 2 * (2 + 3))/10$ ***push***(num, 1.1)
1.1

2. $\mathbf{1}.\mathbf{1} - (5 + 2 * (2 + 3))/10$ ***push***(ops,'-')
1.1
'-'

3. $\mathbf{1}.\mathbf{1} - (\mathbf{5} + 2 * (2 + 3))/10$ ***push***(ops,'('), ***push***(num, 5)
1.1;5
'-';'('

4. $\mathbf{1}.\mathbf{1} - (\mathbf{5} + 2 * (2 + 3))/10$ ***push***(ops,'+')
1.1;5
'-';'(';'+'

5. $\mathbf{1}.\mathbf{1} - (\mathbf{5} + \mathbf{2} * (2 + 3))/10$ ***push***(num, 2)
1.1;5;2
'-';'(';'+'

5. $\mathbf{1}.\mathbf{1} - (\mathbf{5} + \mathbf{2} * (2 + 3))/10$ ***push***(ops, '*')
1.1;5;2
'-';'(';'+';'*'

7. $\mathbf{1}.\mathbf{1} - (\mathbf{5} + \mathbf{2} * (\mathbf{2} + 3))/10$ ***push***(ops, '('), ***push***(num, 2)
1.1;5;2;2
'-';'(';'+';'*'; '('

8. $\mathbf{1}.\mathbf{1} - (\mathbf{5} + \mathbf{2} * (\mathbf{2} + 3))/10$ ***push***(ops, '+')
1.1;5;2;2
'-';'(';'+';'*'; '('; '+'

9. $\mathbf{1}.\mathbf{1} - (\mathbf{5} + \mathbf{2} * (\mathbf{2} + \mathbf{3}))/10$ ***push***(num, 3)
1.1; 5; 2; 2; 3
'-';'(';'+';'*'; '('; '+'

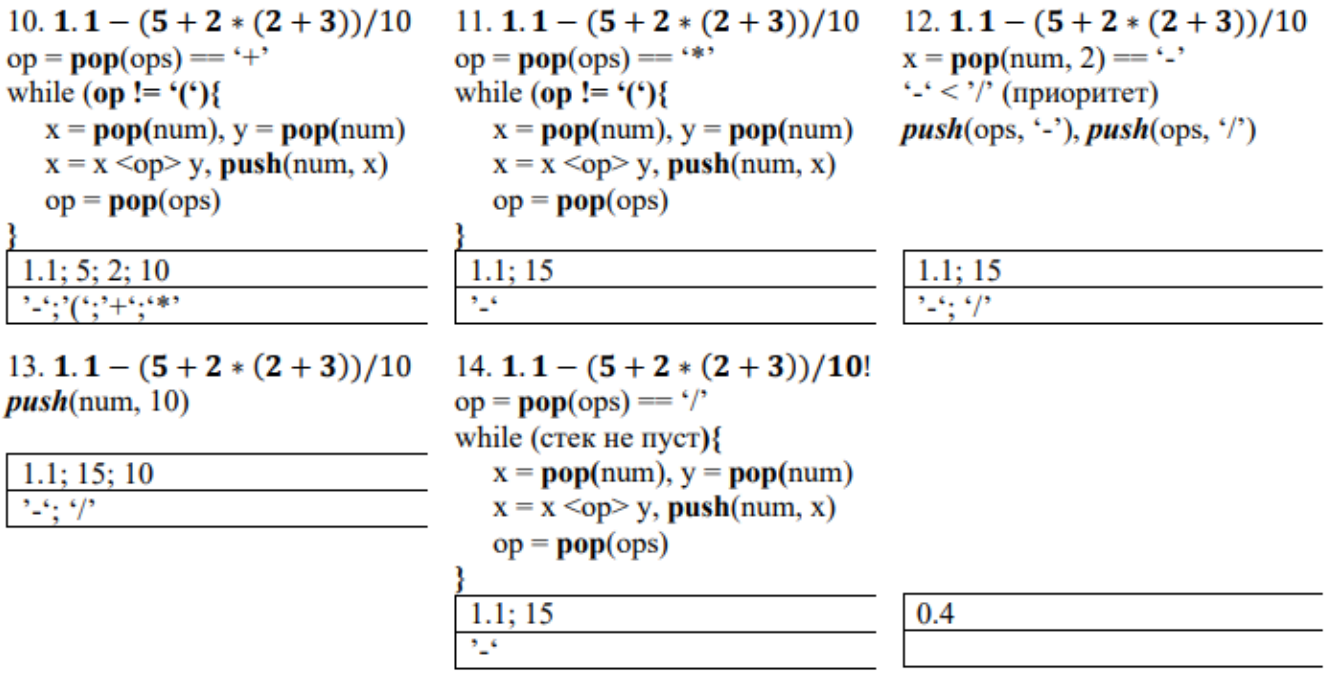

Рисунок 1. Пошаговый алгоритм использования стека

Анализ задачи

1. Для реализации калькулятора необходимо реализовать структуру данных «стек». Стек должен поддерживать операции подавления элемента в стек и извлечения последнего добавленного элемента. Стек будем реализовывать на основе связанного списка, т. к. это позволит иметь адрес следующего элемента. Так же это позволяет производить операции добавления и извлечения за время O(1).
2. Добавление элемента в стек работает по следующему принципу. Каждый новый элемент добавляется на вершину стека и имеет указатель на следующий элемент в стеке. Для добавления элемента в стек устанавливается указатель нового элемента на текущий верхний элемент стека, а затем обновляется указатель верхнего элемента стека, чтобы он указывал на новый элемент.
3. Извлечение минимального элемента из стека происходит следующим образом. Мы извлекаем элемент, находящийся на вершине, при этом на его место встает элемент, находившийся под извлеченным. Тем самым следующий элемент становится вершиной.
4. Для создания стека используется следующий алгоритм:

```
1  struct stack *stack_create()
2  {
3   struct stack *s = malloc(sizeof(*s));
4   if (s != NULL) {
5   s->size = 0;
6   s->top = NULL;
7   }
8   return s;
9  }
```

5. Кроме того, в программе используются две структуры “listnode”, одна для сохранения чисел, а другая для хранения операторов. Обе структуры включают два компонента: “value” для хранения значений и “next” для хранения ссылки на следующий узел.
6. Также нужно реализовать функции для подсчета и расстановки приоритетов при работе со стеками: score(char* s), которая производит вычисления и возвращает окончательные данные, add_stack_sums(char* s, n_stack* num, c_stack* ops), которая разделяет переданные данные между двумя стеками и выполняет предварительное вычисление для сохранения приоритета операций, а также syntax(char *s), которая проверяет корректность входных данных.

Тестовые данные

Протестируем программу используя разные вариации входных данных. Предполагается, что программа будет проверять корректность введенных данных и выдавать отрицательный код ошибки в случае, если данные были введены неправильным образом.

Проверка различных вариантов ввода данных:

```
root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "2+2"
Выражение равно = 4.000000.
root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "(2+2)"
Выражение равно = 4.000000.
```

```
root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "1.5+4.5"
Выражение равно = 6.000000.
root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "(1.5+4.5)"
Выражение равно = 6.000000.
```

```
root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "3+(4*2)"
Выражение равно = 11.000000.
root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "3+4*2"
Выражение равно = 11.000000.
```

```
root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "(3+4)*2"
Выражение равно = 14.000000.
```

```
root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "(3.2+4)*(2+7/3)-4+2"
Выражение равно = 29.200000.
```

Можно заметить, что программа учитывает знаки приоритета и распределяет приоритеты даже по скобкам, а так же воспринимает дробные числа. Ответ во всех случаях верный.

Теперь попробуем несколько вариантов некорректного ввода данных, чтобы удостовериться, что программа выдаст ошибку

```
root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "3++4"
ERROR SYNTAX1!!!
ERROR -3
```

```
root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "(4*7+2(("
ERROR SYNTAX1!!!
ERROR -12
```

```
root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "9.5+5.33.3"
ERROR SYNTAX1!!!
ERROR -2
```

В данных случаях программа выдала ошибку, как и было задумано.

Листинг программы

main.c

```c
#include "score.h"
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int k;
    if ((k = check_brackets(argv[1])) < 0)
    {
```

```
 9          fprintf(stderr,
10                  RED
11                  "ERROR SYNTAX1!!!\n"
12                  "ERROR %d\n" RESET,
13                  k);
14          return -1;
15      }
16      if ((k = syntax(argv[1])) < 0)
17      {
18          fprintf(stderr,
19                  RED
20                  "ERROR SYNTAX1!!!\n"
21                  "ERROR %d\n" RESET,
22                  k);
23          return -1;
24      }
25      else
26      {
27          double number = score(argv[1]);
28          printf("Выражение равно = %f.\n", number);
29      }
30      return 0;
31  }
```

llist_char.c

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  typedef struct c_listnode
 4  {
 5      char c_value;
 6      struct c_listnode *next;
 7  } c_listnode;
 8
 9  c_listnode *c_list_createnode(char value)
10  {
11      c_listnode *p;
12      p = malloc(sizeof(c_listnode));
13      if (p != NULL)
14      {
15          p->c_value = value;
16          p->next = NULL;
17      }
18      return p;
19  }
20
21  c_listnode *c_list_addfront(c_listnode *list, char c_value)
22  {
23      c_listnode *newnode;
24      newnode = c_list_createnode(c_value);
25      if (newnode != NULL)
26      {
27          newnode->next = list;
28          return newnode;
29      }
30      return list;
31  }
```

llist_char.h

```
1  typedef struct c_listnode
2  {
3      char value;
4      struct c_listnode *next;
```

```
5  } c_listnode;
6  c_listnode *c_list_createnode(char value);
7  c_listnode *c_list_addfront(c_listnode *list, char c_value);
```

llist_num.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  typedef struct n_listnode
4  {
5      double n_value;
6      struct n_listnode *next;
7  } n_listnode;
8  n_listnode *n_list_createnode(double value)
9  {
10     n_listnode *p;
11     p = malloc(sizeof(n_listnode));
12     if (p != NULL)
13     {
14         p->n_value = value;
15         p->next = NULL;
16     }
17     return p;
18 }
19 n_listnode *n_list_addfront(n_listnode *list, double n_value)
20 {
21     n_listnode *newnode;
22     newnode = n_list_createnode(n_value);
23     if (newnode != NULL)
24     {
25         newnode->next = list;
26         return newnode;
27     }
28     return list;
29 }
```

llist_num.h

```
1  typedef struct n_listnode
2  {
3      double value;
4      struct n_listnode *next;
5  } n_listnode;
6  n_listnode *n_list_createnode(double value);
7  n_listnode *n_list_addfront(n_listnode *list, double value);
```

score.c

```
1  #include "score.h"
2  #define RESET "\033[0m"
3  #define RED "\033[31m"
4  void priority(n_stack *num, c_stack *ops)
5  {
6      if (ops->size != 0)
7      {
8          if (ops->top->value == '/' || ops->top->value == '*')
9          {
10             double tepm_1 = stack_pop_num(num);
11             double tepm_2 = stack_pop_num(num);
12             stack_pop_char(ops) == '/' ? stack_push_num(num, tepm_2 / tepm_1)
13                                        : stack_push_num(num, tepm_2 * tepm_1);
14         }
15     }
16 }
17 void score_stack(n_stack *num, c_stack *ops)
18 {
```

```
    while (ops->size != 0 && num->size != 0)
    {
        double temp_1 = stack_pop_num(num);
        double temp_2 = stack_pop_num(num);
        stack_pop_char(ops);
        stack_push_num(num, temp_2 + temp_1);
    }
}
int add_ops_stack(char s, c_stack *ops, int *signal)
{
    if (s == '-')
    {
        *(signal) *= -1;
        return stack_push_char(ops, '+');
    }
    else
    {
        return stack_push_char(ops, s);
    }
}
int add_stack_sums(char *s, n_stack *num, c_stack *ops)
{
    unsigned int i = 0;
    int signal = 1;
    while (s[i] != '\n' && s[i] != '\0')
    {
        if (isdigit(s[i]))
        {
            if (stack_push_num(num, atof(s + i) * signal) != 0)
            {
                fprintf(stderr, RED "ERROR ADD IN STACK NUM\n" RESET);
                return -1;
            }
            signal = 1;
            priority(num, ops);
            while (isdigit(s[i]) || s[i] == '.')
                i++;
        }
        else if (s[i] == ' ')
        {
            i++;
        }
        else if (s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/')
        {
            if (add_ops_stack(s[i], ops, &signal) != 0)
            {
                fprintf(stderr, RED "ERROR ADD IN STACK CHAR\n" RESET);
                return -1;
            }
            i++;
        }
        else if (s[i] == '(')
        {
            if (stack_push_num(num, score(s + i + 1) * signal) != 0)
            {
                fprintf(stderr, "ERROR ADD IN STACK NUM\n");
                return -1;
            }
            signal = 1;
            i++;
            int scobka = 1;
```

```
80              while (scobka != 0)
81              {
82                  if (s[i] == '(')
83                      scobka++;
84                  else if (s[i] == ')')
85                      scobka--;
86                  i++;
87              }
88              priority(num, ops);
89          }
90          else if (s[i] == ')')
91          {
92              break;
93          }
94          else
95          {
96              fprintf(stderr, RED "ERROR!!! EXTRANEOUS SYMBOLS\n" RESET);
97              return -1;
98          }
99      }
100     return 0;
101 }
102
103 double score(char *s)
104 {
105     n_stack *num = stack_create_num();
106     c_stack *ops = stack_create_char();
107     if (s == NULL)
108     {
109         fprintf(stderr, "ERROR\n");
110         return 0;
111     }
112     if (add_stack_sums(s, num, ops) < 0)
113     {
114         fprintf(stderr, RED "ERROR IN ADDING STACK!!!\n" RESET);
115         return -1;
116     }
117     double result = 0;
118     score_stack(num, ops);
119     result = num->top->value;
120     stack_free_num(num);
121     stack_free_char(ops);
122     return result;
123 }
```

score.h

```
1  #include <assert.h>
2  #include <ctype.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include "llist_char.h"
6  #include "llist_num.h"
7  #include "ssll_char.h"
8  #include "ssll_num.h"
9  #include "syntax.h"
10 #define RESET "\033[0m"
11 #define RED "\033[31m"
12 void score_stack(n_stack *num, c_stack *ops);
13 void priority(n_stack *num, c_stack *ops);
14 int add_ops_stack(char s, c_stack *ops, int *signal);
15 double score(char *s);
16 int add_stack_sums(char *s, n_stack *num, c_stack *ops);
```

ssl_char.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "llist_char.h"
typedef struct c_stack
{
    c_listnode *top;
    int size;
} c_stack;
c_stack *stack_create_char()
{
    c_stack *s = malloc(sizeof(*s));
    if (s != NULL)
    {
        s->size = 0;
        s->top = NULL;
    }
    return s;
}
char stack_pop_char(c_stack *s)
{
    c_listnode *next;
    char value;
    if (s->top == NULL)
    {
        fprintf(stderr, "Stack underflow");
        return -1;
    }
    next = s->top->next;
    value = s->top->value;
    s->top = next;
    s->size += -1;
    return value;
}
void stack_free_char(c_stack *s)
{
    while (s->size > 0)
    {
        stack_pop_char(s);
    }
    free(s);
}
int stack_size_char(c_stack *s)
{
    return s->size;
}
int stack_push_char(c_stack *s, char value)
{
    s->top = c_list_addfront(s->top, value);
    if (s->top == NULL)
    {
        fprintf(stderr, "Stack overflow\n");
        return -1;
    }
    s->size += 1;
    return 0;
}
```

ssl_char.h

```c
typedef struct c_stack
{
```

```
    c_listnode *top;
    char size;
} c_stack;
c_stack *stack_create_char();
void stack_free_char(c_stack *s);
int stack_size_char(c_stack *s);
int stack_push_char(c_stack *s, char value);
char stack_pop_char(c_stack *s);
```

ssl_num.c

```
#include <stdio.h>
#include <stdlib.h>
#include "llist_num.h"
typedef struct n_stack
{
    n_listnode *top;
    int size;
} n_stack;
n_stack *stack_create_num()
{
    n_stack *s = malloc(sizeof(*s));
    if (s != NULL)
    {
        s->size = 0;
        s->top = NULL;
    }
    return s;
}
double stack_pop_num(n_stack *s)
{
    n_listnode *next;
    double value;
    if (s->top == NULL)
    {
        fprintf(stderr, "Stack underflow");
        return -1;
    }
    next = s->top->next;
    value = s->top->value;
    s->top = next;
    s->size += -1;
    return value;
}
void stack_free_num(n_stack *s)
{
    while (s->size > 0)
    {
        stack_pop_num(s);
    }
    free(s);
}
int stack_size_num(n_stack *s)
{
    return s->size;
}
int stack_push_num(n_stack *s, double value)
{
    s->top = n_list_addfront(s->top, value);
    if (s->top == NULL)
    {
        fprintf(stderr, "Stack overflow\n");
        return -1;
```

```
    }
    s->size += 1;
    return 0;
}
```

ssl_num.h

```
typedef struct n_stack
{
    n_listnode *top;
    int size;
} n_stack;
n_stack *stack_create_num();
void stack_free_num(n_stack *s);
int stack_size_num(n_stack *s);
int stack_push_num(n_stack *s, double value);
double stack_pop_num(n_stack *s);
```

syntax.c

```
#include "syntax.h"
int check_brackets(char *s)
{
    int check_bracket = 0;
    int i = 0;
    while (s[i] != '\n' && s[i] != '\0')
    {
        if (s[i] == '(' && check_bracket >= 0)
        {
            check_bracket++;
        }
        else if (s[i] == ')')
        {
            check_bracket--;
        }
        i++;
    }
    if (check_bracket != 0)
    {
        return -12;
    }
    return 0;
}
int check_numbers(char *s)
{
    int i = 0;
    int check_dot = 0;
    while (isdigit(s[i]) || s[i] == '.')
    {
        if (s[i] == '.')
            check_dot++;
        if (check_dot > 1)
            return -2;
        i++;
    }
    return 0;
}
int syntax(char *s)
{
    int i = 0;
    int num_ops = 0;
    while (s[i] != '\n' && s[i] != '\0')
    {
        if (isdigit(s[i]))
```

```
45          {
46              if (check_numbers(s + i) < 0)
47                  return -2;
48              while (isdigit(s[i]) || s[i] == '.')
49                  i++;
50              num_ops++;
51          }
52          else if (
53              s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/')
54          {
55              num_ops--;
56              if (num_ops < 0)
57                  return -3;
58              i++;
59          }
60          else if (s[i] == '(')
61          {
62              num_ops++;
63              if (syntax(s + i + 1) < 0)
64                  return -4;
65              i++;
66              int bracket = 1;
67              while (bracket != 0)
68              {
69                  if (s[i] == '(')
70                      bracket++;
71                  else if (s[i] == ')')
72                      bracket--;
73                  i++;
74              }
75          }
76          else if (s[i] == ')')
77          {
78              return 0;
79          }
80          else
81          {
82              i++;
83          }
84      }
85      if (num_ops != 1)
86          return -5;
87      return 0;
88  }
```

syntax.h

```
1  #include <ctype.h>
2  #include <stdio.h>
3  int syntax(char *s);
4  int check_brackets(char *s);
```