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 OC GNU/Linux»

Вариант 2.2 «Калькулятор командное строки»

Выполнил:студент гр. ИС-242Денисов 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 А – левый операнд, В – правый операнд, р – арифметическая операция. А и В – представляют собой арифметические выражения или вещественные числа, р = +|-|\*|/. 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!):



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

**Критерии оценки**

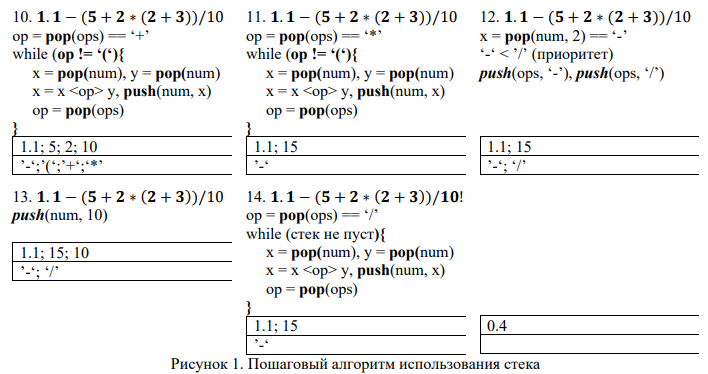
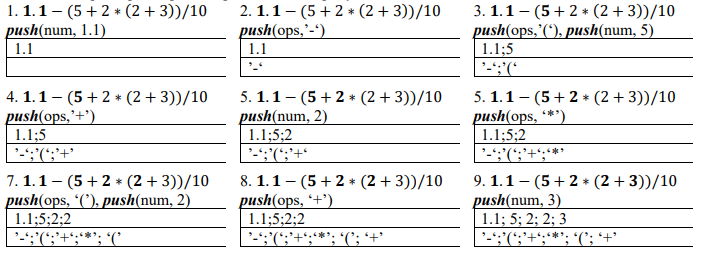
▪ **Оценка «удовлетворительно»:** не предусмотрены скобки и приоритеты операций.

▪ **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 𝟏. 𝟏 − (𝟓 + 𝟐 ∗ (𝟐 + 𝟑))/𝟏𝟎. Пошаговый разбор алгоритма представлен на рисунке 1.



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

1. 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.
2. 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:

|  |  |
| --- | --- |
| 1  2  3  4 | 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. |

|  |  |
| --- | --- |
| 1  2  3  4 | 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. |

|  |  |
| --- | --- |
| 1  2  3  4 | 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. |

|  |  |
| --- | --- |
| 1  2 | root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "(3+4)\*2"  Выражение равно = 14.000000. |

|  |  |
| --- | --- |
| 1  2 | 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

|  |  |
| --- | --- |
| 1  2  3 | root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "3++4"  ERROR SYNTAX1!!!  ERROR -3 |

|  |  |
| --- | --- |
| 1  2  3 | root@DESKTOP-RCKFD8V:~/kursovaya/got/bin# ./calculator "(4\*7+2(("  ERROR SYNTAX1!!!  ERROR -12 |

|  |  |
| --- | --- |
| 1  2  3 | 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**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | #include "score.h"  #include <stdio.h>  #include <stdlib.h>  **int** **main**(**int** argc, **char** \*argv[])  {  **int** k;  **if** ((k = check\_brackets(argv[**1**])) < **0**)  {  fprintf(stderr,  RED  "ERROR SYNTAX1!!!**\n**"  "ERROR %d**\n**" RESET,  k);  **return** -**1**;  }  **if** ((k = syntax(argv[**1**])) < **0**)  {  fprintf(stderr,  RED  "ERROR SYNTAX1!!!**\n**"  "ERROR %d**\n**" RESET,  k);  **return** -**1**;  }  **else**  {  **double** number = score(argv[**1**]);  printf("Выражение равно = %f.**\n**", number);  }  **return** **0**;  } |

**llist\_char.c**

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

**llist\_char.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **typedef** **struct** c\_listnode  {  **char** value;  **struct** c\_listnode \*next;  } c\_listnode;  c\_listnode \***c\_list\_createnode**(**char** value);  c\_listnode \***c\_list\_addfront**(c\_listnode \*list, **char** c\_value); |

**llist\_num.c**

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

**llist\_num.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **typedef** **struct** n\_listnode  {  **double** value;  **struct** n\_listnode \*next;  } n\_listnode;  n\_listnode \***n\_list\_createnode**(**double** value);  n\_listnode \***n\_list\_addfront**(n\_listnode \*list, **double** value); |

**score.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123 | #include "score.h"  #define RESET "\033[0m"  #define RED "\033[31m"  **void** **priority**(n\_stack \*num, c\_stack \*ops)  {  **if** (ops->size != **0**)  {  **if** (ops->top->value == '/' || ops->top->value == '\*')  {  **double** tepm\_1 = stack\_pop\_num(num);  **double** tepm\_2 = stack\_pop\_num(num);  stack\_pop\_char(ops) == '/' ? stack\_push\_num(num, tepm\_2 / tepm\_1)  : stack\_push\_num(num, tepm\_2 \* tepm\_1);  }  }  }  **void** **score\_stack**(n\_stack \*num, c\_stack \*ops)  {  **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**;  **while** (scobka != **0**)  {  **if** (s[i] == '(')  scobka++;  **else** **if** (s[i] == ')')  scobka--;  i++;  }  priority(num, ops);  }  **else** **if** (s[i] == ')')  {  **break**;  }  **else**  {  fprintf(stderr, RED "ERROR!!! EXTRANEOUS SYMBOLS**\n**" RESET);  **return** -**1**;  }  }  **return** **0**;  }  **double** **score**(**char** \*s)  {  n\_stack \*num = stack\_create\_num();  c\_stack \*ops = stack\_create\_char();  **if** (s == NULL)  {  fprintf(stderr, "ERROR**\n**");  **return** **0**;  }  **if** (add\_stack\_sums(s, num, ops) < **0**)  {  fprintf(stderr, RED "ERROR IN ADDING STACK!!!**\n**" RESET);  **return** -**1**;  }  **double** result = **0**;  score\_stack(num, ops);  result = num->top->value;  stack\_free\_num(num);  stack\_free\_char(ops);  **return** result;  } |

**score.h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #include <assert.h>  #include <ctype.h>  #include <stdio.h>  #include <stdlib.h>  #include "llist\_char.h"  #include "llist\_num.h"  #include "ssll\_char.h"  #include "ssll\_num.h"  #include "syntax.h"  #define RESET "\033[0m"  #define RED "\033[31m"  **void** **score\_stack**(n\_stack \*num, c\_stack \*ops);  **void** **priority**(n\_stack \*num, c\_stack \*ops);  **int** **add\_ops\_stack**(**char** s, c\_stack \*ops, **int** \*signal);  **double** **score**(**char** \*s);  **int** **add\_stack\_sums**(**char** \*s, n\_stack \*num, c\_stack \*ops); |

**ssl\_char.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | #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**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **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**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | #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**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **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**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88 | #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]))  {  **if** (check\_numbers(s + i) < **0**)  **return** -**2**;  **while** (isdigit(s[i]) || s[i] == '.')  i++;  num\_ops++;  }  **else** **if** (  s[i] == '+' || s[i] == '-' || s[i] == '\*' || s[i] == '/')  {  num\_ops--;  **if** (num\_ops < **0**)  **return** -**3**;  i++;  }  **else** **if** (s[i] == '(')  {  num\_ops++;  **if** (syntax(s + i + **1**) < **0**)  **return** -**4**;  i++;  **int** bracket = **1**;  **while** (bracket != **0**)  {  **if** (s[i] == '(')  bracket++;  **else** **if** (s[i] == ')')  bracket--;  i++;  }  }  **else** **if** (s[i] == ')')  {  **return** **0**;  }  **else**  {  i++;  }  }  **if** (num\_ops != **1**)  **return** -**5**;  **return** **0**;  } |

**syntax.h**

|  |  |
| --- | --- |
| 1  2  3  4 | #include <ctype.h>  #include <stdio.h>  **int** **syntax**(**char** \*s);  **int** **check\_brackets**(**char** \*s); |