Лабораторная работа №13
Тема: «Обратная польская запись»

Цель работы: сформировать знания и умения по работе с подпрограммами, приобрести навыки написания программ с использованием обратной польской записи (ОПЗ).

Время выполнения: 6 часа.

Теоретические сведения

Преобразование выражения в ОПЗ с использованием стека

Нам понадобится стек для переменных типа char, т.к. исходное выражение мы получаем в виде строки.

Рассматриваем поочередно каждый символ:

1. Если этот символ - число (или переменная), то просто помещаем его в выходную строку.

2. Если символ - знак операции (+, -, *, /), то проверяем приоритет данной операции. Операции умножения и деления имеют наивысший приоритет (допустим он равен 3). Операции сложения и вычитания имеют меньший приоритет (равен 2). Наименьший приоритет (равен 1) имеет открывающая скобка.

Получив один из этих символов, мы должны проверить стек:

а) Если стек все еще пуст, или находящиеся в нем символы (а находится в нем могут только знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек.

б) Если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в выходную строку до тех пор, пока выполняется это условие; затем переходим к пункту а).

3. Если текущий символ - открывающая скобка, то помещаем ее в стек.

4. Если текущий символ - закрывающая скобка, то извлекаем символы из стека в выходную строку до тех пор, пока не встретим в стеке открывающую скобку (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить. Закрывающая скобка также уничтожается.

Если вся входная строка разобрана, а в стеке еще остаются знаки операций, извлекаем их из стека в выходную строку.

Рассмотрим алгоритм на примере простейшего выражения:

Дано выражение:

a + (b - c) * d

Теперь вся входная строка разобрана, но в стеке еще остаются знаки операций, которые мы должны просто извлечь в выходную строку. Поскольку стек - это структура, организованная по принципу LIFO, сначала извлекается символ '*', затем символ '+'.

Итак, мы получили конечный результат:

a b c - d * +

Преобразование выражения в ОПЗ с помощью рекурсивного спуска

Реализация данного алгоритма представляет собой несколько функций, последовательно вызывающих друг друга.

Началом обработки выражения становится вызов функции begin(), которая обрабатывает сложение и вычитание (т.е. сохраняет их во временной переменной и помещает в выходную строку, когда к ней вернется управление после вызова функции mult_div()).

Функция begin() вызывает функцию mult_div(), обрабатывающую знаки деления и умножения (т.е. сохраняет их во временной переменной и помещает в выходную строку, когда к ней вернется управление после вызова функции symbol())..

Далее функция mult_div() вызывает функцию symbol(), обрабатывающую переменные (или числа) и скобки.

Если функция symbol() получает открывающую скобку, она вызывает функцию begin() (т.е. все начинается сначала) и ожидает закрывающей скобки, когда управление вновь возвращается к ней. Если она не дожидаестя закрывающей скобки, это означает, что в выражении содержится синтаксическая ошибка.

Если функция symbol() получает переменную, то помещает ее в выходную строку.

Рассмотрим работу этих функций на примере исходного выражения:

a + (b - c) * d

Передачу управления от одной функции к другой (т.е. вызов функции или возвращение управления вызвавшей функции) обозначим знаком -->

Текущим символом является 'a'. Преобразование начинается с вызова функции begin(). Далее получаем цепочку вызовов begin() --> mult_div() --> symbol(). Функция symbol() помещает символ 'a' в выходную строку, заменяет текущий символ на '+' и возвращает управление. Состояние выходной строки:

a

Текущим символом является '+'. symbol() --> mult_div() --> begin(). Функция begin() запоминает текущий символ во временной переменной и заменяет его на '('. Состояние выходной строки: a

Текущим символом является '('. begin() --> mult_div() --> symbol(). Функция symbol() заменяет текущий символ на 'b' и вызывает функцию begin(). Состояние выходной строки: a

Текущим символом является 'b'. symbol()--> begin() --> mult_div() --> symbol(). Функция symbol() помещает символ 'b' в выходную строку, заменяет текущий символ на '-' и возвращает управление. Состояние выходной строки: a b

Текущим символом является '-'. symbol() --> mult_div() --> begin(). Функция begin() запоминает текущий символ во временной переменной (поскольку эта переменная - локальная, это, конечно, не означает потерю символа '+', сохраненного ранее) и заменяет текущий символ на 'c'. Состояние выходной строки: a b

Текущим символом является 'c'. begin() --> mult_div() --> symbol(). Функция symbol() помещает символ 'c' в выходную строку, заменяет текущий символ на ')' и возвращает управление. Состояние выходной строки: a b c

Текущим символом является ')'. Поскольку закрывающую скобку ни одна функция не обрабатывает, функции поочередно возвращают управление, пока оно не возвратится к функции symbol(), которая обрабатывала открывающую скобку, т.е. цепочка будет такой: symbol() --> mult_div() --> begin(). (здесь функция begin() помещает сохраненный символ '-' в выходную строку) begin() --> symbol(). Далее функция symbol() заменяет текущий символ на '*' Состояние выходной строки: a b c -

Текущим символом является '*'. symbol() --> mult_div() Функция mult_div() запоминает текущий символ во временной переменной и заменяет его на 'd' Состояние выходной строки: a b c -

Текущим символом является 'd'. mult_div() --> symbol(). Функция symbol() помещает символ 'd' в выходную строку и возвращает управление. Состояние выходной строки: a b c - d

Строка разобрана. Возвращение управления: symbol() --> mult_div() (здесь функция mult_div() помещает сохраненный символ '*' в выходную строку). Далее: mult_div() --> begin() (здесь функция begin() помещает сохраненный символ '+' в выходную строку) Состояние выходной строки: a b c - d * +

Реализация на C++ (для работы со строками и исключениями используются классы библиотеки VCL):

```
class TStr2PPN {
```

```
    AnsiString instr, outstr;     //input & output strings
    char curc;                    //the current character
    int iin;                      //the index of the input string

    char nextChar();              //get the next character
    void begin();                 //handle plus & minus
    void mult_div();              //handle multiplication & division
    void symbol();                //handle characters & brackets

  public:
    TStr2PPN() {                  //constructor
      iin = 1;
    }

    void convert(char *str);      //convert to PPN
    AnsiString get_outstr();      //get the output string
};


//get the next character
inline char TStr2PPN::nextChar() {
  if(iin <= instr.Length()) return curc = instr[iin++];
  return curc = '\0';
}

//get the output string
inline AnsiString TStr2PPN::get_outstr(){return outstr;}

//convert to PPN
void TStr2PPN::convert(char *str) {
  try {
    instr = str;
    outstr.SetLength(0);
    iin = 1;
    nextChar();
    //begin the convertation
    begin();
    if(iin != (instr.Length()+1) || curc != '\0') {
      throw Exception("Syntax error");
    }
```

```
      if(!isalpha(instr[instr.Length()]) && instr[instr.Length()]!=')') {
        throw Exception("Syntax error");
      }
    }
    catch(...) {throw;}
}

//handle plus & minus
void TStr2PPN::begin() {
    try {
       mult_div();
       while(curc=='+' || curc=='-') {
          char temp = curc;
          nextChar();
          mult_div();
          outstr += temp;
       }
    }
    catch(...) {throw;}
}

//handle multiplication & division
void TStr2PPN::mult_div() {
    try {
       symbol();
       while(curc=='*' || curc=='/') {
          char temp = curc;
          nextChar();
          symbol();
          outstr += temp;
       }
    }
    catch(...) {throw;}
}

//handle characters
void TStr2PPN::symbol() {
    try {
      if(curc=='(') {
        nextChar();
```

```
        begin();
        if(curc!=')') {
          throw Exception("Error: wrong number of brackets");
        }
        else nextChar();
      }
      else
        if(isalpha(curc)) {
          outstr += curc;
          nextChar();
        }
        else {throw Exception("Syntax error");}
    }
    catch(...) {throw;}
}
```

Алгоритм вычисления выражения, записанного в ОПЗ

Для реализации этого алгоритма используется стек для чисел (или для переменных, если они встречаются в исходном выражении). Алгоритм очень прост. В качестве входной строки мы теперь рассматриваем выражение, записанное в ОПЗ:

1. Если очередной символ входной строки - число, то кладем его в стек.

2. Если очередной символ - знак операции, то извлекаем из стека два верхних числа, используем их в качестве операндов для этой операции, затем кладем результат обратно в стек.

Когда вся входная строка будет разобрана в стеке должно остаться одно число, которое и будет результатом данного выражения.

Рассмотрим этот алгоритм на примере выражения:

7 5 2 - 4 * +

```
int calculate(string str_in) {
  Stack<int> val_stack;              //стек
  int n1, n2, res;

  for(i = 0; i<str_in.length(); ++i) {
    if(isNumber(str_out[i])) {
      val_stack.push(str_out[i]);
    }
    else {
      n2 = val_stack.pop();
```

```
      n1 = val_stack.pop();
      switch(str_out[i]) {
        case '+': res = n1 + n2; break;
        case '-': res = n1 - n2; break;
        case '*': res = n1 * n2; break;
        case '/': res = n1 / n2; break;
        default: cout<<"Ошибка !\n";
      }
      val_stack.push(res);
    }
   }
  return val_stack.pop();
}
```

Общие задания к лабораторной работе №13

1. Постфиксной формой записи (ОПЗ) выражения a◦b называется запись, в которой знак операции размещен за операндами ab◦.

Например:

Обычная запись Обратная польская запись

a-b a b -

a*b+c a b * c +

a*(b+c) a b c + *

(a+c)/(c*a-d) a c + c a * d - /

Описать функции, которая вычисляет значение заданного выражения.

Входные данные. В первой строке содержит обратную польскую запись арифметического выражения. Все операнды целые положительные числа.

Выходные данные. Вывести результат вычисления ОПЗ.

Технические требования. Используются знаки операций: +, -, *, /.

Примеры

input.txt output.txt

3 1 + 4

12 5 * 10 - 50

1 2 30 + * 32

2 10 + 2 4 + 6 – 2 / 6

2. На вход программы поступает выражение, состоящее из односимвольных идентификаторов и знаков арифметических действий. Требуется преобразовать это выражение в обратную польскую запись или же сообщить об ошибке.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретическую часть лабораторной работы.

2. Реализовать индивидуальное задание по вариантам, представленные в теоретических сведениях, сделать скриншоты работающих программ. Написать комментарии.

3. Написать отчет, содержащий:

1. Титульный лист, на котором указывается:

а) полное наименование министерства образование и название учебного заведения;

б) название дисциплины;

в) номер практического занятия;

г) фамилия преподавателя, ведущего занятие;

д) фамилия, имя и номер группы студента;

е) год выполнения лабораторной работы.

2. Индивидуальное задание из раздела «Теоретические сведения» с кодом, комментариями и скриншотами работающих программ.

3. Построение блок-схем.