Лабораторная работа №7.

Основы лексического анализа.

1. Цель работы

Знакомство с основами лексического анализа программного кода на языке C++. Получение практических навыков разработки программ на языке C++ с использованием стандартного класса string.

2. Основные сведения

Элементы лексического анализа.

Лексический анализатор (или сканер) – это часть компилятора, которая читает исходную программу и выделяет в ее тексте лексемы входного языка. На вход лексического анализатора поступает текст исходной программы. Цель лексического анализа при трансляции состоит в переводе текста исходной программы к стандартному виду и преобразование его во внутренний язык компилятора, когда все компоненты текста (имена, числа, знаки операций) или слова будут иметь одинаковый формат, что облегчает дальнейшую обработку программы.

Лексемой (словом) является элементарная конструкция языка, рассматриваемая в данной программе как неделимый символ, имеющий определённый смысл. Лексемами языков программирования являются знаки операций (`+`, `-`, `*`, `/`, `;`, `%`, `=`, `>`, `<`), комбинации знаков (`!=`, `<=`, `>=`, `==`, `(*`, `->`), зарезервированные слова (`for`, `do`, `not`, `int` и т.д.), идентификаторы или имена (`Sum`, `P1`, `Laba2`, `Found`, `Count`), числа (3, 7.05E-1, -3.5 и т.д.).

Друг от друга слова могут разделяться пробелом (пробелами), концами строк и комментариями. Кроме того, разделителями могут быть знаки, например, `Sum=Sum+X;`. Здесь словами являются: `Sum`, `=`, `+`, `X`, `;`. Слова `=`, `+`, `;` служат также и разделителями. Если пробел встретится между символами слова, то возникает ошибка или два новых слова. Например, число 7.08E15 пробелом после цифры 8 разбивается на число 7.08 и имя E15 и теряет свой смысл.

Например, зарезервированное слово состоит из символов латинских букв и ограничивается слева символом, отличным от латинской буквы, а справа – любым символом, не являющимся латинской буквой, и принадлежит алфавиту языка. Именем является последовательность букв и цифр, начинающаяся с буквы или символа подчёркивания. Имя слева ограничено не буквой, а справа – символом, не являющимся буквой или цифрой.

Выделение границ лексем является нетривиальной задачей. Ведь в тексте исходной программы лексемы никак не ограничены. Примером может служить оператор языка C , имеющий вид `k=i+++++j;`. Существует только одна единственно верная трактовка этого оператора: `k = i++ + ++j;` (равносильная конструкции вида `k = (i++) + (++j);`). Однако найти ее лексический анализатор может, лишь просмотрев весь оператор до конца и перебрав все варианты (например, вариант `k = (i++)++ + j;` является синтаксически правильным, но семантикой языка C не допускается).

В основном лексические анализаторы «обнаруживают» в тексте исходной программы комментарии и пробельные символы (пробелы, табуляцию, перевод строки), а также выделяют лексемы следующих типов: идентификаторы, строковые, символьные и числовые константы, ключевые (служебные) слова входного языка, знаки операций и разделители.

Результатом работы лексического анализатора является перечень всех найденных в тексте исходной программы лексем. Этот перечень лексем можно представить в виде таблицы, называемой *таблицей лексем*. Здесь каждой лексеме соответствует некий уникальный условный код, зависящий от её типа, и дополнительная служебная информация. Кроме того, информация о некоторых типах лексем, найденных в исходной программе, должна помещаться в *таблицу идентификаторов* (или в одну из таблиц идентификаторов, если компилятор предусматривает различные таблицы идентификаторов для различных типов лексем)[1].

При лексическом анализе производится и лексический контроль – выявление недопустимых и неописанных слов (ошибка типа «неизвестный идентификатор»).

Класс string.

Как известно, язык C++ не предусматривает встроенного типа для строк. Для работы с ними существует несколько возможностей:

[1] Молчанов, А.Ю. Системное программное обеспечение: уч. для вузов, 3-е изд. – СПб.: Питер, 2012. – 400 с.

1) традиционный массив символов, завершающийся нулевым байтом (C-строка): `char [N];`

2) объект класса `string` (включён в библиотеку C++, начиная со стандарта ISO/ANSI C++98): `std::basic_string`.

Фактически класс `string` является специализацией более общего шаблонного класса `basic_string`, который обобщает способы управления и хранения последовательностей символов. Создание, манипулирование и уничтожение строк осуществляется удобным набором методов класса и связанных функций.

Для работы с классом `string` в программе должен быть включен заголовочный файл `string`. Класс `string` является частью пространства имен `std`, поэтому вы должны указать директиву `using` либо ссылаться на класс как `std::string`. Определение класса скрывает природу строки как массива символов и позволяет трактовать ее как обычную переменную.

Перечислим основные возможности по работе с данными типа string:

Создание и удаление:

конструктор класса – создает или копирует строку;

деструктор – уничтожает строку.

Размер и ёмкость:

capacity() – возвращает количество символов, которое строка может хранить без дополнительного выделения памяти (ёмкость строки);

empty() – возвращает логическое значение, указывающее, является ли строка пустой;

length(), size() – возвращают количество символов в строке;

max_size() – возвращает максимальный размер строки, который может быть выделен;

resize() – расширяет или уменьшает строку (удаляет или добавляет символы в конце строки);

reserve() – расширяет или уменьшает ёмкость строки;

shrink_to_fit() – изменяет размер строки так, чтобы её длина стала равной её ёмкости.

Доступ к элементам:

`[]`, at() – доступ к элементу по заданному индексу;

back(), front() - доступ к последнему и первому элементу строки.

Изменение:

=, assign() – присваивают новое значение строке;

+=, append(), push_back() – добавляют символы к концу строки;

insert() – вставляет символы в произвольную позицию строки;

clear() – удаляет все символы строки;

erase() – удаляет символы по произвольному индексу строки;

pop_back() – удаляет последний элемент строки;

replace() – заменяет символы произвольных индексов строки другими символами;

swap() – меняет местами значения двух строк.

Ввод/вывод:

>>, getline() – считывают значения из входного потока в строку;

<< – записывает значение строки в выходной поток;

c_str(), data() – конвертирует строку в строку C-style с нуль-терминатором в конце;

copy() – копирует содержимое строки (которое без нуль-терминатора) в массив типа char.

Сравнение строк:

==, !=, <, <=, >, >= – сравнивают строки (возвращают значение типа bool);

compare() – сравнивает, являются ли две строки равными/неравными (возвращает `-1`, `0` или `1`).

Подстроки и конкатенация:

+ – соединяет две строки;

substr() – возвращает подстроку.

Поиск:

find – ищет индекс первого символа/подстроки;

find_first_of – ищет индекс первого символа из набора символов;

find_first_not_of – ищет индекс первого символа НЕ из набора символов;

find_last_of – ищет индекс последнего символа из набора символов;

find_last_not_of – ищет индекс последнего символа НЕ из набора символов;

rfind – ищет индекс последнего символа/подстроки.

Поддержка итераторов[2]*:*

begin(), end() – возвращают «прямой» итератор, указывающий на первый и последний (элемент, который идет за последним) элементы строки;

rbegin(), rend() – возвращают «обратный» итератор, указывающий на последний (т.е. «обратное» начало) и первый (элемент, который предшествует первому элементу строки – «обратный» конец) элементы строки. По сравнению с begin() и end() движение итераторов происходит в обратную сторону.

3. Задание

Разработать часть лексического анализатора исходного кода программы на языке C++ с использованием возможностей библиотеки string.

Исходные данные должны считываться из текстового файла построчно. Работа с данными должна быть организованна с помощью строк string. Результаты обработки должны сохраняться в файле (изменённый файл, таблица лексем и т.д.). Для проверки правильности обработки данных необходимо продумать удобный формат вывода на экран процесса формирования нового файла (например, исходная строка и результат её обработки по образцу из л.р. №8 за 1 семестр «Текстовые файлы»).

В качестве входного файла берутся только cpp-коды. Реализовать возможность ввода имени входного и выходного файлов как через командную строку программы, так и с помощью консольного ввода. Реализовать использование ключей при работе с командной строкой:

`-I` для указания имени файла для анализа. Например, `-I  file1.cpp`;

`-O` для указания выходного файла. Например, `-O  file2.txt`.

В процессе разработки лексического анализатора необходимо учитывать правила анализа кода программы компилятором: слова, похожие на лексемы языка, являющиеся частью строковых литералов, однострочных и многострочных комментариев, других лексем, лексемами НЕ являются и, как следствие, не обрабатываются и НЕ включаются в таблицы лексем и идентификаторов.

4. Индивидуальные задания

Индивидуальное задание выбирается в соответствии с порядковым номером студента в журнале преподавателя. Варианты для реализации:

1. Удалить «пустые» строки, лишние пробельные символы (пробел, табуляция) и указать удалённое количество каждого их них в конце строки как однострочный комментарий. После удаления должен остаться только один пробельный символ в группе. Пример оформления комментария:
   `Prob-5, Tab-0, Enter - 3.`
2. Перенести в конец файла содержимое всех однострочных комментариев с указанием номера строки, в которой они используются. Например, `стр. 1: // комментарий 1`.
3. Переставить в комментарий в конце строки вещественные типы данных, указав в скобках их количество. Например, строка
   ```
   float a="float type"; double d=1;float b;
   ```
   приводится к виду
   ```
   a="float type";  d=1;b; // float(2) double(1)
   ```
4. Записать ключевые слова циклов (for, while, do-while) заглавными буквами и подсчитать вид каждого из них.
5. Записать названия целочисленных типов данных в объявлениях переменных, функций, прототипов функций, структур заглавными буквами.
6. Отобразить на экране количество многострочных комментариев в коде и удалить их.
7. Замена текста строковых литералов их длиной. Например, строковый литерал `"string"` должен быть заменён на литерал `"6"`. Отобразить на экране информацию о количестве сделанных замен.
8. Замена всех символьных литералов их ASCII-кодами. Отобразить в порядке возрастания ASCII-кодов на экране информацию о количестве сделанных замен в виде:
   `ASCII-код – литерал – количество замен`
   Например,
   ```
   10 - '\n' - 6
   97 - 'a'  - 4
   ```

[2] Итератор – структура данных, которая позволяет получить доступ к определённому элементу контейнера STL (string, map, vector, list, …). (Галовиц, Я. C++17 STL. Стандартная библиотека шаблонов. – СПб.: Питер, 2018. – 432 с.)

9. Составить таблицу слов, являющихся целыми числами:

№	Число	Встречается в строках

Например,

№	Число	Встречается в строках
1	-8	1, 56
2	24523	1
3	0	34, 56

10. Составить таблицу слов, являющихся строковыми литералами:

№	Литерал	Встречается в строках

Например,

№	Литерал	Встречается в строках
1	"\n"	1, 56
2	"Hello"	1
3	"0"	34, 56

11. Составить таблицу идентификаторов, включающих цифры:

№	Идентификатор	Встречается в строках

Например,

№	Идентификатор	Встречается в строках
1	N8	1, 56
2	color_45	1
3	from1to4	34, 56

12. Все числа в формате с фиксированной запятой оформить многострочным комментарием.

13. Добавление в конец строки информации в виде однострочного комментария о присутствующих в ней операторах сравнения с указанием их количества: <, <=, >, >=. Пример оформления комментария: `<(2), <=(0), >(0), >=(0)`. Учесть, что символ < может быть частью директивы `#include`, оператора вставки в поток <<, а символ > – частью директивы `#include`, оператора косвенного членства `->`, оператора извлечения из потока >>.

14. Добавление в конец строки информации в виде однострочного комментария о присутствующих в выражениях знаках арифметических операций с указанием их количества: +, *, +=, *=. Пример оформления комментария: `+(2), *(1), +=(0), *=(0)`. Учесть, что символ + может быть частью оператора ++, а символ * – частью оператора разыменовывания (`(*a).b`, `*a+1`) и многострочного комментария (`/*`, `*/`).

15. Перестановка в конец строки всех выводимых с помощью cout выражений с указанием их порядкового номера. Например, строка

```
cout<< "a" <<(a<1);
```

преобразуется к виду

```
cout<<  <<; // 1."a" 2.(a<1)
```

16. Преобразовать все выводимые выражения в строковые литералы. Например, строка

```
cout<<a  << (a<1);
```

преобразуется к виду

```
cout<<"a"  << "(a<1)";
```

4. Подготовка

1. Изучить теоретические сведение по стандартному классу string. Изучить библиотечные функции для работы со строками и выбрать необходимые для решения поставленной задачи.

Рекомендованные ресурсы по работе со строками:

https://en.cppreference.com/w/cpp/string/basic_string

https://www.cplusplus.com/reference/string/string/

2. Разработать тестовый cpp-файл для полной проверки разрабатываемой программы по своему варианту. Записать предполагаемый результат обработки этого файла. Включить тестовые данные, которые программа не обрабатывает или обрабатывает не корректно.

3. Разработать блок-схему алгоритма решения задачи: последовательность вызовов подпрограмм.

5. Состав отчета

1. Индивидуальное задание.
2. Подробное словесное описание алгоритма обработки данных.
3. Описание используемых функций класса string: заголовок, описание формальных параметров и возвращаемого значения. В качестве образца для описания функций взять справочник https://en.cppreference.com/w/cpp/string/basic_string.
4. Тестовые данные, которые программа не обрабатывает или обрабатывает не корректно.
5. Затраты времени на каждое действие по разработке и выполнению программы с указанием числа строк программы, реализующих это действие. Лучше в форме таблицы с итогом: общие затраты времени и общее число строк программы. Указать производительность труда в строках в час.

6. Пример программы

Упрощенная версия игры "Палач". Программа хранит список слов в массиве объектов string, выбирает одно слово случайным образом и предлагает игроку угадать буквы в слове. Шесть неудачных попыток означают проигрыш. В программе используются:

- функция find() для проверки попыток,
- функция lengtth() для получения длины строки,
- различные конструкторы создания объектов string,
- оператор [] для доступа к буквам,
- операция != для сравнения строк,
- операция += для создания объекта string, в котором хранятся неудачные попытки.

Для отслеживания удачных попыток, в программе создастся слово такой же длины, что и загаданное, но состоящее из дефисов. При удачной попытке дефис заменяется угаданной буквой.

```
// hangman.cpp – игра "Палач"
#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>
#include <cctype>

using std::string;
const int NUM = 26;
const string wordlist[NUM] = {"apiary", "beetle", "cereal",
"danger", "ensign", "florid", "garage", "health", "insult",
"jackal", "keeper", "loaner", "manage", "nonce", "onset",
"plaid", "quilt", "remote", "stolid", "train", "useful",
"valid", "whence", "xenon", "yearn", "zippy"};

int main()
{
      using std::cout;
      using std::cin;
      using std::tolower;
      using std::endl;
      std::srand(std::time(0));
      char play;
      cout << "Will you play a word game? <y/n> "; // запуск игры
      cin >> play;
      play = tolower(play);
      while (play == 'y') {
            // создание инициализированного объекта строки
            string target = wordlist[std::rand() % NUM];
            int length = target.length();
            // создаём строку длиной length из повторяющихся символов '-'
            string attempt(length, '-');
            // создаём пустую строку
            string badchars;
            int guesses = 6;
            // выводим информацию о загаданном слове
            cout << "Guess my secret word. It has " << length
                  << " letters, and you guess\n"
```

```
            << "one letter at a time. You get " << guesses
            << " wrong guesses.\n";
      cout << "Your word: " << attempt << endl; //  вывод слова
      while (guesses > 0 && attempt != target) {
            char letter;
            cout << "Guess a letter: ";
            cin >> letter;
            // проверка введённой буквы
            if (badchars.find(letter) != string::npos
              || attempt.find(letter) != string::npos) {
                  cout << "You already guessed that. Try again.\n" ;
                  continue;
            }
            int loc = target.find(letter);
            // используем string::npos для признак отсутствия
            // введённой буквы
            if (loc == string::npos) {
                  cout << "Oh, bad guess !\n";
                  --guesses;
                  badchars += letter; // добавить к строке
            } else {
                  cout << "Good guess!\n";
                  attempt[loc]=letter;
                  // Проверить, не появляется ли буква еще раз
                  loc = target. find (letter, loc + 1) ;
                  while (loc != string::npos) {
                        attempt[loc]=letter;
                        loc = target.find (letter, loc + 1) ;
                  }
            }
            cout << "Your word: " << attempt << endl;
            if (attempt != target) {
                  if (badchars.length() > 0)
                        cout << "Bad choices: " << badchars << endl;
                  cout << guesses  << "bad guesses left\n" ;
            }
      }
      if (guesses > 0)
            cout << "That's right!\n";
      else
            cout << "Sorry, the word is " << target << " \n" ;
      cout << "Will you play another? <y/n> " ;
      cin >> play;
      play = tolower(play);
      }
cout << "Bye\n";
return 0;
}
```