**Реализация на таблици с последователен достъп. Работа с файлове.**

***Задача:***

1. *Да се напише клас, който да описва характеристиките на една лексическа единица: идентификатор,*

*тип,*

*дескриптор,*

*адрес.*

*Класът да притежава и следните методи:*

*подходящи конструктори;*

*operator=;*

*метод, който отпечатва полетата на класа.*

1. *Да се напише клас, който да описва и обработва таблица на дескриптори и притежава:*

* *Таблица на дескриптори, която е с 4 полета (таблица с последователен достъп) и съхранява информация за лексически единици.*

|  |  |  |  |
| --- | --- | --- | --- |
| идентификатор | тип | дескриптор | адрес |
| i | int | V1 | 0 |
| *delta* | *float* | V2 | 2 |
| *varch* | *char* | V3 | 6 |
| … |  |  |  |

*Фигура 1*

* *броя на лексическите единици;*
* *конструктор;*

*методи за:*

* *търсене на лексическа единица, по зададен идентификатор;*
* *добавяне на лексическа единица (в края на таблицата), като дескриптора и адреса се изчисляват спрямо предходната (за улеснение се приема, че за тип char се заделя 1 байт, за int – 2 байта, за float - 4 байта, за double - 8 байта);*
* *премахване на лексическа единица, като се извърши последващо уплътняване на таблицата;*
* *отпечатване на таблицата.*

*Да се състави програма, която да чете лексически единици от текстов файл в следния формат:*

*<тип> <идентификатор>;*

*<тип> <идентификатор>;*

*<тип> <идентификатор>;*

*...*

*(например:*

*int i;*

*double delta;*

*char varchar;*

*...)*

*Програмата да отпечатва таблицата на класа.*

***Упътване и указанния за работа:***

Знаете, че при деклариране на променлива в дадена програма, за нея се заделя памет (в байтове) в зависимост от нейния тип. Чрез идентификаторите (имената на променливите) в програмата можем да се обръщаме към тази памет, за да съхраняваме и обработваме различни стойности. За всяка програма се създава таблица на дескрипторите. Една такава таблица, доста упростена, е предмет на следващия проект. Тя ще съдържа само променливи от основните типове в С++ (виж *Фигура 1*).

Целта на следващите 4 упражнения е да реализираме различни алгоритми за търсене, затова и таблицата е представена в този опростен вид. Полетата *идентификатор* и *тип* нямат нужда от пояснения. Полето *дескриптор* ще го разглеждаме като служебно име, което се генерира автоматично, по реда на постъпване на лексическата единица. Стойността му е комбинация от буква и пореден номер. Полето *адрес* ще показва адреса на първия байт от паметта, заделена за тази променлива. За улеснение ще започва от 0. За да може дескриптора и адреса за всяка променлива да се определя автоматично и коректно, предлагам да се декларират допълнително две статични променливи към класа, описващ лексическата единица (Потърсете допълнителна информация за статични променливи!). Пояснявам само, че статичната променлива е обща за всички обекти от този клас, нейната стойност може да се ползва и променя както от всеки един обект, така и директно от класа. Статичната променлива не се инициализира в конструктора, а като глобална променлива.

Пример за клас, описващ лексическа единица:

class lexicalUnit

{

char identificator[20];

char type[20];

char descriptor[5];

int address;

void setAddress(char \*type); // метод, който задава адреса и генерира следващия свободен адрес, според типа,който има тази лексическа единица

void setDescriptor(); // метод, който задава дескриптора и генерира следващия

public:

static int newAddress; // статична променлива, указваща адреса, на който да се запише текущата единица

static int newDescriptor; // статична променлива, указваща текущия дескриптор

lexicalUnit();

lexicalUnit(char \*pIdentificator, char \*pType);

lexicalUnit& operator=(lexicalUnit& lu);

~lexicalUnit();

char\* getIdentificator(); // метод, който връща идентификатора

void print();

};

Методите getIdentificator и operator= са помощни и се ползват в другия клас.

За да преобразувате число в стринг може да използвате функцията sprintf или itoa.

Пример за използване и промяна на статична променлива от обект:

void lexicalUnit::setAddress(char \*type)

{

address = newAddress; //полето на класа получава стойността на статичната променлива, след което тя се променя според типа

int size=0;

if (strcmp(type, "int") == 0)

size = 2;

if (strcmp(type, "char") == 0)

size = 1;

if (strcmp(type, "float") == 0)

size = 4;

if (strcmp(type, "double") == 0)

size = 8;

newAddress += size;

}

Пример за клас, съдържащ таблицата на дескрипторите с последоватален достъп:

class descriptors

{

lexicalUnit table[100];

int number;

public:

descriptors();

~descriptors();

int add(lexicalUnit &lu); // връща 1, при успешно добавяне и 0 в противен случай

int del(char \*pIdentificator); // връща 1, при успешно изтриване и 0 в противен случай

int search(char \*pIdentificator); // връща позицията на лексическата единица в масива или -1 в противен случай

void print();

};

Когато дефинирате методите, имайте предвид че:

1. Лексическа единица не може да се добави ако няма място, ако е подаден невалиден тип, ако вече има добавена с такъв идентификатор.
2. Лексическа единица не може да се премахне ако таблицата е празна или ако няма такава.
3. Методът за търсене би ви помогнал да определите има ли такава (ако не, може да я добавите) и къде се намира (за да я премахнете).

Пример за инициализация на статична променлива:

#include "stdafx.h"

#include <iostream>

#include "descriptors.h"

#include "lexicalUnit.h"

#include <stdio.h>

#include <stdlib.h>

using namespace std;

int lexicalUnit::newAddress = 0; // инициализация на статична променлива

int lexicalUnit::newDescriptor = 1; // инициализация на статична променлива

int \_tmain(int argc, \_TCHAR\* argv[])

{

descriptors D;

.......

return 0;

}

Може да ползвате дадените примери или да си разпишете свои класове. Свободни сте да реализирате задачата по ваше усмотрение. Успешна работа.