# Dictionary cpp

// Подключение предварительно скомпилированных заголовков

#include "stdafx.h"

#include "macros.h"

// Основная функция программы

int main()

{

// Установка локали для корректного отображения символов на русском языке

setlocale(LC\_ALL, "ru");

// Установка кодировки консоли для ввода

SetConsoleCP(1251);

// Установка кодировки консоли для вывода

SetConsoleOutputCP(1251);

// Объявление переменных для выбора пользователем действий, размера идентификатора и имени, а также массивов словаря и записей

int choice, size = 0, size\_num = 1, id = 0;

char name[DICTNAMEMAXSIZE];

Dictionary::Instance new\_dictionary[DICTMAXSIZE];

Dictionary::Entry inst[ENTRYNAMEMAXSIZE];

// Вывод меню для пользователя

Dictionary::menu();

// Запрос выбора действия у пользователя

std::cout << "Выберите действие: ";

std::cin >> choice;

// Цикл обработки выбора действий пользователя

do

{

try

{

// Обработка выбора действия пользователя с использованием switch

switch (choice)

{

case 1:

// Создание нового словаря и увеличение размера

new\_dictionary[size] = Dictionary::Create(name, size);

size++;

break;

case 2:

// Добавление новой записи в словарь и увеличение размера

Dictionary::AddEnrty(new\_dictionary, inst, size\_num);

size\_num++;

break;

case 3:

// Удаление записи из словаря

Dictionary::DelEntry(new\_dictionary, id, size\_num);

break;

case 4:

// Получение записи из словаря

Dictionary::GetEntry(new\_dictionary, id, size\_num);

break;

case 5:

// Обновление записи в словаре

Dictionary::UpdEntry(new\_dictionary, inst, size\_num);

std::cout << "Элемент изменен!!!" << std::endl;

break;

case 6:

// Удаление всех элементов словаря

Dictionary::Delete(new\_dictionary);

std::cout << "Все элементы словаря удалены" << std::endl;

break;

case 7:

// Вывод всех элементов словаря

Dictionary::Print(new\_dictionary, size);

break;

}

}

catch (const std::exception& e)

{

// Обработка исключений и вывод сообщения об ошибке

std::cout << e.what() << std::endl;

}

// Запрос выбора действия у пользователя в конце каждой итерации цикла

std::cout << "Выберите действие: ";

std::cin >> choice;

} while (choice != 0);

// Тестирование словаря с помощью макроса TEST\_DICTIONARY

TEST\_DICTIONARY;

return 0;

}

# Dictionary h

// Подключение стандартных заголовочных файлов

#include <cstring>

#include <iostream>

// Определение максимального размера имени словаря

#define DICTNAMEMAXSIZE 20

// Определение максимального размера словаря

#define DICTMAXSIZE 10

// Определение максимального размера имени записи в словаре

#define ENTRYNAMEMAXSIZE 30

// Определение текстовых сообщений для исключений

#define THROW01 "Create: превышен размер имени словаря"

#define THROW02 "Create: превышен размер максимальной емкости словаря"

#define THROW03 "AddEntry: переполнение словаря"

#define THROW04 "AddEntry: дублирование идентификатора"

#define THROW05 "GetEntry: не найден элемент"

#define THROW06 "DelEntry: не найден элемент"

#define THROW07 "UpdEntry: не найден элемент"

#define THROW08 "UpdEntry: дублирование идентификатора"

// Объявление пространства имен Dictionary

namespace Dictionary

{

// Структура записи в словаре

struct Entry

{

int id;

char name[ENTRYNAMEMAXSIZE];

};

// Структура экземпляра словаря

struct Instance

{

char name[DICTNAMEMAXSIZE];

int maxsize = 0;

Entry\* dictionary;

};

// Объявление функций для работы со словарем

void menu();

Instance Create(char name[DICTNAMEMAXSIZE], int size);

void AddEnrty(Instance inst[], Entry ed[], int size\_num);

void DelEntry(Instance inst[], int id, int size\_num);

void UpdEntry(Instance inst[], Entry new\_ed[], int size\_num);

Entry GetEntry(Instance inst[], int id, int size\_num);

void Print(Instance d[], int size);

void Delete(Instance d[]);

}

// Проверка наличия более одного макроса для тестирования словаря

#if defined(TEST\_DICTIONARY) + defined(TEST\_CREATE\_01) + defined(TEST\_CREATE\_02) + defined(TEST\_ADDENTRY\_03) + defined(TEST\_ADDENTRY\_04) + defined(TEST\_GETENTRY\_05) + defined(TEST\_DEL\_ENTRY\_06) + defined(TEST\_UPD\_ENTRY\_07) + defined(TEST\_UPD\_ENTRY\_08) > 1

// Выдача ошибки компиляции в случае наличия более одного макроса для тестирования словаря

#error "Можно использовать только один макрос из списка"

#endif

# Function\_and\_macros cpp

// Подключение предварительно скомпилированного заголовочного файла

#include "stdafx.h"

// Подключение заголовочного файла с макросами

#include "macros.h"

// Отключение предупреждения компилятора 4996 (о deprecated функциях)

#pragma warning(disable : 4996)

// Определение функции меню словаря

void Dictionary::menu()

{

// Вывод пунктов меню на экран

std::cout << "1 - Создать экземпляр словаря " << std::endl;

std::cout << "2 - Добавить элемент словаря " << std::endl;

std::cout << "3 - Удалить элемент словаря " << std::endl;

std::cout << "4 - Получить элемент словаря " << std::endl;

std::cout << "5 - Изменить элемент словаря " << std::endl;

std::cout << "6 - Удалить элементы словаря " << std::endl;

std::cout << "7 - Распечатать элементы словаря " << std::endl;

std::cout << "0 - Выход " << std::endl << std::endl;

}

// Определение функции создания экземпляра словаря

Dictionary::Instance Dictionary::Create(char name[DICTNAMEMAXSIZE], int size)

{

// Создание нового экземпляра словаря

Instance new\_dictionary;

// Создание фейкового экземпляра словаря (для возврата в случае ошибки)

Instance fake;

// Выделение памяти для массива записей словаря

new\_dictionary.dictionary = new Entry[DICTMAXSIZE];

// Ввод названия нового словаря

std::cout << "Введите название нового словаря: ";

std::cin >> name;

// Копирование названия в структуру нового словаря

strcpy(new\_dictionary.name, name);

// Проверка на превышение максимального размера имени словаря

if (strlen(name) > DICTNAMEMAXSIZE)

{

throw std::length\_error(THROW01);

return fake;

}

// Проверка на превышение максимальной емкости словаря

if (size >= DICTMAXSIZE)

{

throw std::overflow\_error(THROW02);

return fake;

}

// Возврат созданного экземпляра словаря

return new\_dictionary;

}

// Определение функции добавления записи в словарь

void Dictionary::AddEnrty(Instance inst[], Entry ed[], int size\_num)

{

int i;

// Выбор номера словаря для добавления записи

std::cout << "Выберите для какого словаря вы хотите вписать элементы: ";

std::cin >> i;

i--;

// Проверка на допустимость номера словаря

if (i < 0 || i >= size\_num)

{

std::cout << "Недопустимый номер словаря" << std::endl;

return;

}

// Проверка на превышение максимальной емкости словаря

if (inst[i].maxsize >= DICTMAXSIZE)

{

throw std::exception(THROW03);

return;

}

// Ввод данных для новой записи

std::cout << "Введите имя: ";

std::cin >> inst[i].dictionary[inst[i].maxsize].name;

std::cout << "Введите id: ";

std::cin >> inst[i].dictionary[inst[i].maxsize].id;

// Проверка на дублирование идентификатора

for (int j = 0; j < inst[i].maxsize; j++)

{

if (inst[i].dictionary[j].id == inst[i].dictionary[inst[i].maxsize].id)

throw std::exception(THROW04);

}

// Увеличение счетчика записей в словаре

inst[i].maxsize++;

}

// Определение функции изменения записи в словаре

void Dictionary::UpdEntry(Instance inst[], Entry new\_ed[], int size\_num)

{

int i, choice\_number;

// Выбор номера словаря и номера записи для изменения

std::cout << "Выберите номер словаря, элемент которого вы хотите изменить: ";

std::cin >> i;

i--;

std::cout << "Выберите номер элемента: ";

std::cin >> choice\_number;

choice\_number--;

// Проверка на допустимость номера элемента

if (choice\_number < 1 && choice\_number > inst[i].maxsize)

{

throw std::exception(THROW07);

return;

}

// Проверка на допустимость номера словаря

if (i < 0 || i >= size\_num)

{

std::cout << "Недопустимый номер словаря" << std::endl;

return;

}

int new\_id;

bool id\_exists = false;

// Ввод нового идентификатора

std::cout << "Введите id: ";

std::cin >> new\_id;

// Проверка на дублирование идентификатора

for (int j = 0; j < inst[i].maxsize; j++)

{

if (j != choice\_number && inst[i].dictionary[j].id == new\_id)

{

id\_exists = true;

throw std::overflow\_error(THROW08);

return;

break;

}

}

// Изменение идентификатора записи

if (!id\_exists)

inst[i].dictionary[choice\_number].id = new\_id;

}

// Определение функции получения записи из словаря по идентификатору

Dictionary::Entry Dictionary::GetEntry(Instance inst[], int id, int size\_num)

{

Entry element;

int i;

// Выбор номера словаря для поиска записи

std::cout << "Выберите номер словаря, элемент которого вы хотите найти: ";

std::cin >> i;

i--;

// Проверка на допустимость номера словаря

if (i < 0 || i >= size\_num)

{

std::cout << "Недопустимый номер словаря" << std::endl;

return element;

}

// Ввод идентификатора для поиска записи

std::cout << "Введите id для поиска элемента: ";

std::cin >> id;

// Поиск записи по идентификатору

for (int j = 0; j < inst[i].maxsize; j++)

{

if (inst[i].dictionary[j].id == id)

{

// Вывод найденной записи

std::cout << inst[i].dictionary[j].name << std::endl;

std::cout << inst[i].dictionary[j].id << std::endl << std::endl;

return element;

}

}

// В случае, если запись не найдена, генерация исключения

throw std::exception(THROW05);

}

// Определение функции печати всех записей в словаре

void Dictionary::Print(Instance d[], int size)

{

// Перебор всех словарей

for (int i = 0; i < size; i++)

{

// Вывод названия словаря

std::cout << "------------ " << d[i].name << " ------------ " << std::endl;

// Перебор всех записей в словаре

for (int j = 0; j < d[i].maxsize; j++)

{

std::cout << d[i].dictionary[j].name << std::endl;

std::cout << d[i].dictionary[j].id << std::endl << std::endl;

}

}

}

// Определение функции удаления записи из словаря

void Dictionary::DelEntry(Instance inst[], int id, int size\_num)

{

int i;

// Выбор номера словаря для удаления записи

std::cout << "Выберите для какого словаря вы хотите вписать элементы: ";

std::cin >> i;

i--;

// Проверка на допустимость номера словаря

if (i < 0 || i >= size\_num)

{

std::cout << "Недопустимый номер словаря" << std::endl;

return;

}

// Проверка на превышение максимальной емкости словаря

if (inst[i].maxsize >= DICTMAXSIZE)

{

std::cout << "Слишком много элементов в словаре" << std::endl;

return;

}

// Ввод идентификатора для удаления записи

std::cout << "Введите id для удаления элемента: ";

std::cin >> id;

// Поиск записи по идентификатору и удаление ее

for (int j = 0; j < inst[i].maxsize; j++)

{

if (inst[i].dictionary[j].id == id)

{

inst[i].dictionary[j] = inst[i].dictionary[inst[i].maxsize - 1];

inst[i].maxsize--;

std::cout << "Элемент успешно удален" << std::endl;

return;

}

}

// В случае, если запись не найдена, генерация исключения

throw std::exception(THROW06);

}

// Определение функции удаления словаря

void Dictionary::Delete(Instance d[])

{

int i;

// Выбор номера словаря для удаления

std::cout << "Выберите номер словаря для удаления: ";

std::cin >> i;

i--;

// Проверка на допустимость номера словаря

if (i < 0 || i >= DICTMAXSIZE)

{

std::cout << "Недопустимый номер словаря" << std::endl;

return;

}

// Очистка памяти, выделенной под массив записей словаря

delete[] d[i].dictionary;

// Установка указателя на nullptr для избежания ошибок при дальнейшем использовании

d[i].dictionary = nullptr;

// Установка максимального размера словаря в 0

d[i].maxsize = 0;

}

# Stdafx h

// Проверка наличия макроса STDAFX\_H и начало условной компиляции

#ifndef STDAFX\_H

#define STDAFX\_H

// Подключение заголовочного файла Dictionary.h

#include "Dictionary.h"

// Подключение заголовочного файла iostream для работы с потоками ввода-вывода

#include <iostream>

// Подключение заголовочного файла windows.h для работы с функциями Windows API

#include <windows.h>

// Подключение заголовочного файла string для работы со строками C++

#include <string>

// Подключение заголовочного файла cstring для работы со строками C

#include <cstring>

// Определение макросов для тестирования

#define TEST\_CREATE\_01

#define TEST\_CREATE\_02

#define TEST\_ADDENTRY\_03

#define TEST\_ADDENTRY\_04

#define TEST\_GETENTRY\_05

#define TEST\_DEL\_ENTRY\_06

#define TEST\_UPD\_ENTRY\_07

#define TEST\_UPD\_ENTRY\_08

#define TEST\_DICTIONARY

// Окончание условной компиляции

#endif

1. Препроцессор в C++ - это часть компилятора, которая работает перед фактическим компилированием кода. Его задача - обработка и изменение исходного кода до его передачи компилятору для компиляции.

2. Директивы препроцессора включают:

- `#define`: определение макросов.

- `#include`: включение содержимого файла в программу.

- `#undef`: удаление определения макроса.

- `#ifdef`: проверка наличия определения макроса.

- `#ifndef`: проверка отсутствия определения макроса.

- `#if`, `#elif`, `#else`, `#endif`: условная компиляция.

- `#pragma`: специфичные для компилятора инструкции.

- `#error`: генерация ошибки в процессе компиляции.

3. Операторы препроцессора `#` (стрингификация) и `##` (конкатенация) используются для манипуляции с макросами:

- Стрингификация (`#`): превращает аргумент макроса в строковый литерал.

Пример:

```cpp

#define STRINGIFY(x) #x

std::cout << STRINGIFY(hello); // Выведет "hello"

```

- Конкатенация (`##`): объединяет два или более токена в один.

Пример:

```cpp

#define CONCAT(x, y) x ## y

CONCAT(var, 1) = 10; // Переменная var1 получит значение 10

```

4. В C++ используются три типа памяти:

- Статическая память: используется для глобальных переменных и статических переменных.

- Память из кучи (heap): используется для динамически выделенной памяти с помощью операторов `new` и `delete`.

- Память из стека: используется для локальных переменных и параметров функций.

5. Пространство имен в C++ - это механизм, который позволяет группировать идентификаторы в определенную область видимости, чтобы предотвратить конфликты имен.

6. Исключение - это механизм обработки ошибок в C++, который позволяет программе передавать управление из блока кода, где произошла ошибка, к блоку кода, который способен ее обработать.

7. Принцип связи инструкций `throw` и `catch` при обработке исключения заключается в том, что `throw` используется для генерации исключения, а `catch` - для его обработки. Если исключение не было обработано в блоке `try`, оно распространяется по стеку вызова функций до тех пор, пока не будет найден блок `catch`, который сможет его обработать.

8. Выражение "необработанное в функции исключение распространяется по стеку вызова функций" означает, что если исключение не обработано в текущем блоке `try-catch`, оно будет передано на уровень выше по стеку вызова функций, где будет снова попытаться обработаться.

Cd название

Cd..

Lib /LIST название.lib