# Лабораторная работа 6

**Ввод, сортировка и двоичный поиск в массиве структур.**

Англо-русский словарь построен в виде массива структур Dictionary и хранится в файле. Структура содержит английское слово и соответствующее ему русское слово. Максимальный размер словаря – 100 пар слов.

Разработать программу, которая:

* обеспечивает формирование словаря (добавление и удаление записей);
* записывает словарь, отсортированный по английским значениям слов, в файл;
* обеспечивает просмотр словаря;
* выполняет перевод слов с английского на русский, используя для поиска слова в словаре метод двоичного поиска в отсортированном массиве;
* выполняет перевод слов с русского на английский, используя для поиска слов в словаре метод перебора;

Программа должна обеспечивать диалог с помощью меню.

Начальное число слов в словаре равно 10.

Планируемое время выполнения работы - 6 часов.

**Шаги разработки программы**

1. *Определение состава и способа представления исходных данных, результатов и промежуточных данных.*

Исходные данные. Так как максимальный размер словаря по условию задачи ограничен, а фактическое число записей в словаре может изменяться , то для хранения словаря в оперативной памяти (ОП) можно использовать нединамический массив из 100 элементов типа Dictionary.

const int max\_size = 100;

const int l\_word = 31;

struct Dictionary {

char engl[l\_word]; // слово по-английски

char rus[l\_word]; // слово по-русски

};

Максимальная длина русских и английских слов *l\_word* = 30 символам. Словарь хранится в текстовом файле. Элементы массива структур Dictionary будем записывать в файл последовательно, начиная с нулевого. При этом поля структуры записываются в виде отдельных строк, т.е. каждое слово должно заканчиваться символом ‘\0’.

1. *Разработка алгоритма решения задачи.*

При программировании этой задачи уделим дополнительное внимание разбиению на функции и спецификации их интерфейсов. Например, логично оформить в виде функции каждую операцию со словарем (формирование, поиск, добавление и уда­ление элемента), поскольку они представляют собой законченные действия.

Интерфейс пользователя организуем в виде простейшего меню, которое будет выводиться на экран после каждого действия. В стандарт C++ не входят функции для работы с экраном в графическом режиме, поскольку они зависят от операционной системы. Поэтому меню представим в виде пронумерованного перечня возможных действий пользователя, красиво размещенного на экране, а выбор действия будем выполнять путем ввода его номера в перечне. В функции *menu()* следует предусмотреть реакцию на ввод пользователем непредусмотренных алгоритмом данных, например, буквы вместо номера (так называемая «защита от дурака»).

Будем исходить из того, что все функции должны быть независимы, чтобы изме­нения в одной функции не могли влиять на поведение другой. Для этого всё, что функциям необходимо получать извне, будем передавать им через параметры (за исключением нединамического массива структур *Dictionary*, который будет глобальной переменной, так как он используется всеми функциями программы).

Прежде всего определим интерфейс нашей программы. В соответствии с заданием, кажется логичным предоставить пользователю следующие возможности:

1. добавление слов в словарь;
2. удаление слов из словаря;
3. перевод слов с английского на русский;
4. перевод слов с русского на английский;
5. просмотр словаря (вывод на экран словаря из ОП);
6. вывод словаря в файл;
7. выход.

Каждый пункт этого меню, кроме последнего, оформим в виде отдельной функ­ции. Определим прототипы предложенных функций.

*Меню*. Эта функция (в качестве подсказки) выводит пронумерованный перечень возможных действий пользователя и выполняет ввод номера выбранного действия. Это число она должна вернуть в вызвавшую функцию.

int Menu( );

*Добавление* *слов в словарь.* Чтобы добавить элемент Dictionary в словарь, надо задать пару слов и определить место, куда вставлять элемент, чтобы массив оставался отсортированным. Поиск места вставки и ввод добавляемого элемента выполним внутри функции, а в функцию передадим указатель на начало словаряи его размерность. Назовем ее addWord.

void addWord(Dictionary \* dict, int & n);

По аналогии определите прототипы остальных функций самостоятельно.

1. *Кодирование и тестовые примеры.*

В этой работе удобно использовать технологию создания программы «сверху вниз»: сначала отладить главную функцию, а затем постепенно добавлять к ней остальные. На месте еще не добавленных функций обычно ставятся так на­зываемые *заглушки* — функции, единственный действием которых является вы­вод сообщения о том, что эта функция была вызвана. Первой добавляемой функцией должна быть функция Menu.

При использовании меню главная функция сильно упрощается: она периодически вызывает меню, анализирует запрошенный код операции и вызывает функцию, которая выполняет эту операцию. При выборе операции с номером 7 выполнение программы завершается.

int main(){

Dictionary dict[100]; //массив структур для хранения словаря в оперативной памяти

int num\_w=0; //фактическое число записей в словаре

while (true) {

switch (menu()) {

case 1:add\_w(dict, num\_w); break;

case 2:

case 3:

case 4: break;

case 5: print\_dict(dict,num\_w); break;

case 6: break;

case 7: return 0;

default: cout<<" Надо вводить число от 1 до 7"<<endl; break;

}

}

return 0;

}

После отладки Menu(), запрограммируйте и отладьте функции add\_w() и print\_dict().

add\_w() добавляет записи в массив структур, а print\_dict() распечатывает массив из оперативной памяти.

### Приложение 1

СТРУКТУРЫ

Структуры в C++ обладают практически теми же возможностями, что и классы, но чаще их применяют просто для логического объединения связанных между со­бой данных. В структуру, в противоположность массиву, можно объединять дан­ные различных типов.

Например, требуется обрабатывать информацию о расписании работы конференц-зала, и для каждого мероприятия надо знать время, тему, фамилию организатора и количество участников. Поскольку вся эта информация относится к одному собы­тию, логично дать ему имя, чтобы впоследствии можно было к нему обращаться. Для этого описывается новый тип данных (обратите внимание на то, что после описания стоит точка с запятой):

struct Event {

int hour, min;

char theme[100], name[100];

int num;

};

Имя этого типа данных — Event. Можно описать переменные этого типа точно так же, как переменные встроенных типов, например:

Event el, e2[10]; // структура и массив структур

Переменные структурного типа можно размещать и в динамической области па­мяти, для этого надо описать указатель на структуру и выделить под нее место:

Event \*pe = new Event; // структура

Event \*pm = new Event[m]: // массив структур

Элементы структуры называются *полями.* Поля могут быть любого основного типа, массивом, указателем, объединением или структурой. Для обращения к полю ис­пользуется *операция выбора* («точка» (.) для переменной и -> для указателя), на­пример:

el.hour = 12; el.min= 30;

strncpy(e2[0].theme,"Выращивание кактусов ", 99);

ре-> num = 30; // или (\*pe).num = 30;

pm[2].hour = 14; // или (\*(pm + 2)).hour = 14;

Структуры одного типа можно присваивать друг другу:

\*ре = el; pm[l] = el; pm[4] = e2[0];

Но присваивание - это и все, что можно делать со структурами целиком. Другие операции, например сравнение на равенство или вывод, не определены. Впрочем, пользователь может задать их самостоятельно, поскольку структура является ви­дом класса, а в классах можно определять собственные операции. Мы рассмотрим эту тему во втором семестре.

*Ввод/вывод структур,* как и массивов, выполняется поэлементно. Вот, например, как выглядит ввод и вывод описанной выше структуры e1:

cin >> e1.hour >> e1.min;

cin. getline (e1. theme, 100);

cout << e1.hour << ' ' << e1.min << ' ' << e1.theme << endl;

Структуры (но не динамические) можно инициализировать перечисле­нием значений их элементов:

Event е3 = {12, 30, "Выращивание кактусов ", "Петрова",25};