

Лекция 3.1. Структуры (записи), таблицы, строки, множества.

Записи

Запись (структура) - это совокупность данных разного типа состоящая из фиксированного числа компонентов, называемых полями записи. Запись явл-ся одной из структур данных (СД), создаваемых самим пользователем для представления разнородной, но логически связанной информации. Доступ к полю записи осуществляется с помощью его имени в записи.

Дескриптор записи может содержать имя записи, количество полей, их имена, а также указатели значений элементов. Поле записи занимает в памяти непрерывную область, т.е. в дескрипторе достаточно иметь один указатель области значений полей записи, а в описании полей - смещение относительно . начала области. Смещение вычисляется при компиляции программы, что повышает эффективность доступа к полям записи.

Определение структурного типа

На языке Модула-2:

```
1 | type complex = record
2 |     re: real;
3 |     im: real;
4 | end;
```

На языке C :

```
1 | struct complex {
2 |     float re;
3 |     float im;
4 | }
```

Без описания типа

На Си - определение переменных:

```

1 struct {
2     float re;
3     float im;
4 } x, y;
5
6 struct {
7     float r;
8     float i;
9 } z;

```

Переменные `x`, `y`, `z` - один тип

- ▶ Тогда можно объявить переменную `x` комплексного типа:
 - ▶ (`var x: complex;` или `struct complex x;`)
- ▶ и обращаться (логически) к действительной и мнимой частям `x` с помощью конструкции `x.re` (или `x.im` соответственно). Физически - расчет адреса поля.
- ▶ Т.к. размер составного значения структурного типа точно специфицирован, допускается присваивание таких значений, а также функции, вырабатывающие структурные значения.

Запись с вариантами

- ▶ При реальном программировании возникает желание по-разному интерпретировать содержимое одной и той же области памяти в зависимости от конкретных обстоятельств.
- ▶ Вариантная часть м.б. одна, описывается последней, м.б. вложенной. Объём (Vop) опр-ся по максимальному полю. Не контролируется компилятором!
- ▶ В языках Си существует более слабый механизм, но эквивалентный по возможностям, он называется смесью (union). Фактически

Фактически смесь - это запись с вариантами, но без явно поддерживаемого дискриминанта.

Определения структурного типа со смесью в языке

```

1 enum typeauto {truck, pass_car}
2 struct {
3     int tonn;
4     char ing;
5 } truck;
6 struct
7 {
8     int pass;
9     char color[15];

```

```

10 } pass_car;
11 struct
12 {
13     char tr_mark[15], model[15];
14     unsigned int year;
15     typeauto tpa;
16     union
17     {
18         struct pass_car;
19         struct truck;
20     } choice;
21 } auto;

```

- **Плюсы** использования вариантной части: экономия памяти
- **Минус** - вариантная часть не контролируется компилятором, за этим должен следить *программист!*

Лучше компоновать поля структуры начиная с более длинных данных, затем - короткие. Компилятор часто выравнивает память полей, поэтому `sizeof(структура)` может дать большее значение, чем подсчет вручную по кол-ву памяти, занимаемой полями.

Операции над структурой - только поиск по полю.

Таблица

Таблица - это конечное множество записей (элементов таблицы), имеющих одну и ту же организацию. Обычно в таблице для всех элементов одно из полей отводится под хранение ключа, используемого для доступа к соответствующему элементу, ключи можно использовать для обработки записей в таблице, что уменьшает время доступа. Создается дополнит. массив, кот. содержит ключи и индекс записи в основной таблице. Работа идет с массивом ключей, а не с массивом записей.

Плюсы использования таблиц ключей:

- Сокращение времени обработки, но требует дополнительной памяти.
- Разделение понятия хранения данных и их структурирования.
- Таблицы широко используют в компиляторах для создания таблиц операций, идентификаторов, ошибок и т.д. и т.п.
- Принцип разделения хранения и структурирования инф-ции часто применяется при хранении больших объемов информации в файлах (изображения: создаются реестры, содержащие ссылки на изображения, играющие роль ключевого значения и имя файла; массивы записей: это доп-ные массивы, содержащие пары

ключ-расположение).

Подсчет времени выполнения цикла:

► Си

```
1  #include <time.h>
2  clock_t start, stop;
3  double duration1 = -1.0;
4  start = clock();
5  for (int i=0; i<=1000; i++)
6  {
7      // some code
8  }
9  stop = clock();
10 duration1=(double)(stop-start)/CLOCKS_PER_SEC;
```

► Паскаль:

```
1 | DecodeTime (const DateTime:TDateTime; var Hour:Word; varMin:Word; var Sec: W
```

Операции над таблицами:

- Аналогичные операциям над массивами:
- Вставка
 - По номеру
 - По значению поля
- Удаление
 - По номеру
 - По значению поля
- поиск по значению поля (если часто используется, то лучше предварительно отсортировать)

Лабораторная работа № 2. Записи с вариантами, обработка таблиц

Цель работы - приобрести навыки работы с типом данных «запись» («структура»), содержащим вариантную часть, и с данными, хранящимися в таблицах. Оценить относительную эффективность программы (в процентах) по времени и по используемому объему памяти в зависимости от используемого алгоритма и от

объема сортируемой информации

- ▶ Создать таблицу, содержащую не менее 40-ка записей (тип - запись с вариантами). Упорядочить данные в ней по возрастанию ключей, где ключ - любое невариантное поле (по выбору программиста), используя:

1. саму таблицу,
2. массив ключей (возможность добавления и удаления записей в ручном режиме обязательна)

Строки

Строка - это линейно упорядоченная последовательность символов, принадлежащих конечному множеству символов, называемому алфавитом.

Строки обладают следующими важными свойствами:

- ▶ их длина, как правило, переменна, хотя алфавит фиксирован;
- ▶ обычно обращение к символам строки идет с какого-нибудь одного конца последовательности, т.е важна упорядоченность этой последовательности, а не ее индексация; в связи с этим свойством строки часто называют также цепочками;
- ▶ чаще всего целью доступа к строке является на отдельный ее элемент (хотя это тоже не исключается), а некоторая цепочка символов в строке.

Строки - это одномерные массивы символов, которые обычно представлены в оперативной памяти (ОП) двумя способами:

- ▶ в виде структур переменной длины с фиксированным максимумом (тип `string` в Паскале);
- ▶ в виде цепочки символов произвольной длины, заканчивающихся символом нулевого байта #0 (тип `Pchar` в BP 7.0, тип `string` в Delphi,
- ▶ в C - строка - это символьный массив, заканчив. нулевым байтом.

Недостаток строк: если строка становится больше отведенной длины то она усекается.

дескриптор строки

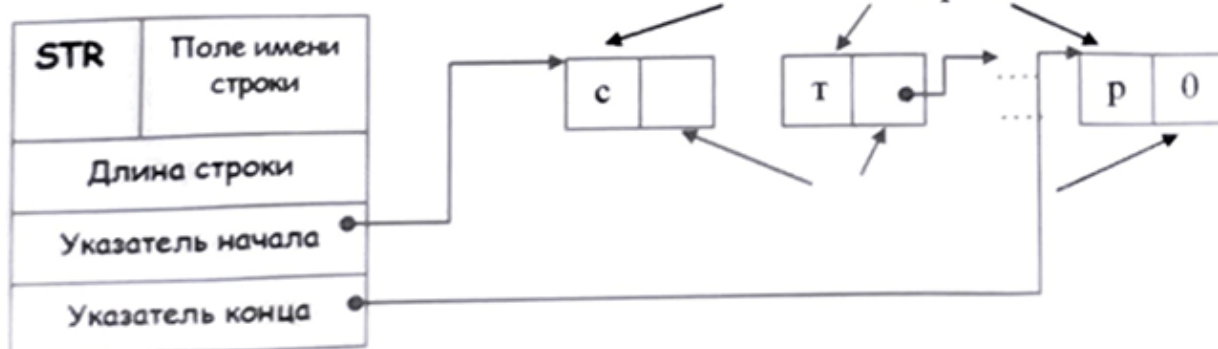
STR	Поле имени строки
Длина строки (6)	

Максимальная
длина (12)
указатель

свободная память



Дескриптор



- ▶ В Windows широко используются так называемые нуль- терминальные строки, максимальная длина кот. - доступная для программы имеющаяся ОП, кот. выделяется на этапе выполнения программы динамически по мере надобности. При компиляции выделяется 4 байта под указатель на начало строки. При выполнении программы, начиная с этого адреса, будет размещаться строка. Важную роль при этом играет счетчик ссылок на строку, кот. размещается в строке за терминальным нулем и занимает 4 байта. С помощью этого счетчика ссылок реализуется “кэширование” памяти, т. е., при копировании строки память не выделяется, в новую переменную помещается содержимое указателя на строку, а счетчик ссылок увеличивается на 1. Т. о., указатели ссылаются на одну и ту же область, счетчик ссылок становится равным двум.
- ▶ Освобождение памяти под длинную строку происходит, когда счетчик ссылок = нулю.

Возможное представление строк в памяти:

ВЕКТОРНОЕ ПРЕДСТАВЛЕНИЕ СТРОК

В дескрипторе: Имя, макс, длина, указатель на начало

- ▶ Представление строк вектором переменной длины с признаком конца (EOS).
- ▶ Представление строк вектором переменной длины со счетчиком. (Pascal) „
- ▶ Вектор с управляемой длиной

Символьно-связное представление строк

- ▶ Однонаправленный линейный список
 - ▶ Двухнаправленный линейный список
 - ▶ Многосимвольные звенья фиксированной длины;
 - ▶ Многосимвольные звенья переменной длины;
 - ▶ Многосимвольные звенья с управляемой длиной
- В дескрипторе - имя, счетчик, указатель начала, указатель конца

Операции над строками

Базовыми операциями над строками являются:

- ▶ определение длины строки;
- ▶ присваивание строк;
- ▶ конкатенация (сцепление) строк;
- ▶ выделение подстроки;
- ▶ поиск вхождения (элемента, группы).

Возникает проблема превышения отведенной памяти в тех языках, где длина строки ограничивается.

Возможны 3 варианта решения, определяемые правилами языка или режимами компиляции:

- ▶ никак не контролировать такое превышение, возникновение такой ситуации неминуемо приводит к трудно локализуемой ошибке при выполнении программы;
- ▶ завершать программу аварийно с локализацией и диагностикой ошибки;
- ▶ ограничивать длину результата в соответствии с объемом отведенной памяти.

Рекомендация: проверять синтаксис вызываемых функций, чтобы убедиться, что параметры действительно являются символами, а не строками. Операция конкатенации, работает только со строками. Поэтому, если необходимо в цикле строку + с символом, то, напр., предварительно задать строку, присвоить ей отдельный символ, далее - сцеплять. Т.к. иначе компилятору придется преобразовывать все символы в строки каждый раз (!), что увеличивает время! Использование любых ф-ций обработки строк увеличивает время выполнения операций!

Множества

Множества - это неупорядоченный набор различных данных одного типа

(порядкового простого), количество которых меняется от 0 до 255. В памяти ПК множество наилучшим образом представляется его характеристической функцией

$$C(S_i) = (i \text{ INS})(i \in s)$$

в виде массива логических значений, i -ая компонента которого обозначает наличие или отсутствие i -ого значения базового элемента во множестве. Размер массива равен числу элементов базового типа во множестве.

Например, множество целых чисел $[1, 3, 6, 9]$ можно представляется в виде следующей последовательности логических значений:

$C(S) = (F \ T \ F \ T \ F \ F \ T \ F \ F \ T)$, F - false, T - true, в диапазон значений $(S) = (0..9)$.

Представление множеств их характеристической функцией позволяет легко реализовать операции: объединения (&), пересечения (+) и разности (-) множеств с помощью элементарных логических операций. Причем, проверка наличия элемента во множестве идет значительно эффективнее при такой реализации: $(i \text{ in } S_1, S_2, \dots, S_n)$, т. к. здесь идет сдвиг и проверка разряда, чем проверка: $IF (i = s_1) \text{ or } (i = s_2) \text{ or } \dots \text{ or } (i = s_n) \dots$

Можно представлять множества, как и строки, связным списком, где элементы множества - это элементы списка. В этом случае список м. б. и сортированный, где элементы множества связаны отношением " $<$ " и тогда при нахождении конкретного элемента нет необходимости просматривать весь список (множество). Множества следует использовать, если базовые типы "небольшие", тогда эффективность этой структуры гарантирована.