Лекция 9.

СЛОЖНЫЕ СТРУКТУРЫ ДАННЫХ НА ОСНОВЕ УКАЗАТЕЛЕЙ (продолжение)

Динамические структуры

Создание массивов во время выполнения программы по сравнению с этапом компиляции намного выгоднее с точки зрения рационального использования памяти. То же самое касается и структур.

①Указатели на структуры объявляются так же, как и указатели на любые другие типы данных, с помощью символа *****, стоящего перед именем экземпляра структуры. Например, указатель `p_addr` на структуру `addr` объявляется так:

```
struct addr {
   int price;
};
struct addr * p_addr;  // объявление в стиле C
addr * p_addr;  // объявление в стиле C++ -
                // не обязательно указывать ключевое слово struct.
```

②Инструментом для выделения пространства под структуру, опять-таки, послужит операция `new`. С ее помощью можно создавать *динамические структуры*. Динамические здесь снова означает выделение памяти во время выполнения, а не во время компиляции.

Например, чтобы выделить память для хранения данных структуры `addr` и присвоить ее адрес соответствующему указателю, можно поступить следующим образом:

```
addr * p_addr = new addr;
```

Это присвоит указателю `p_addr` адрес участка памяти достаточного размера, чтобы вместить данные типа `addr`. Обратите внимание, что синтаксис операции `new` в точности такой же, как и для встроенных типов C++.

③Более сложная часть — *доступ к членам* структуры. Когда вы создаете динамическую структуру, то не можете применить операцию членства к имени структуры (оператор точка), поскольку эта структура безымянна. Все, что у вас есть — ее адрес. В C++ предусмотрена специальная операция для этой ситуации — *операция членства через указатель* (**->**). Эта операция, оформленная в виде тире с последующим значком "больше", означает для указателей на структуры то же самое, что операция точки для имен структур.

Например, `p_addr->price` означает член `price` структуры, на которую указывает `p_addr`. Ещё пример обращения к полям динамической структуры показан на рис. 3.

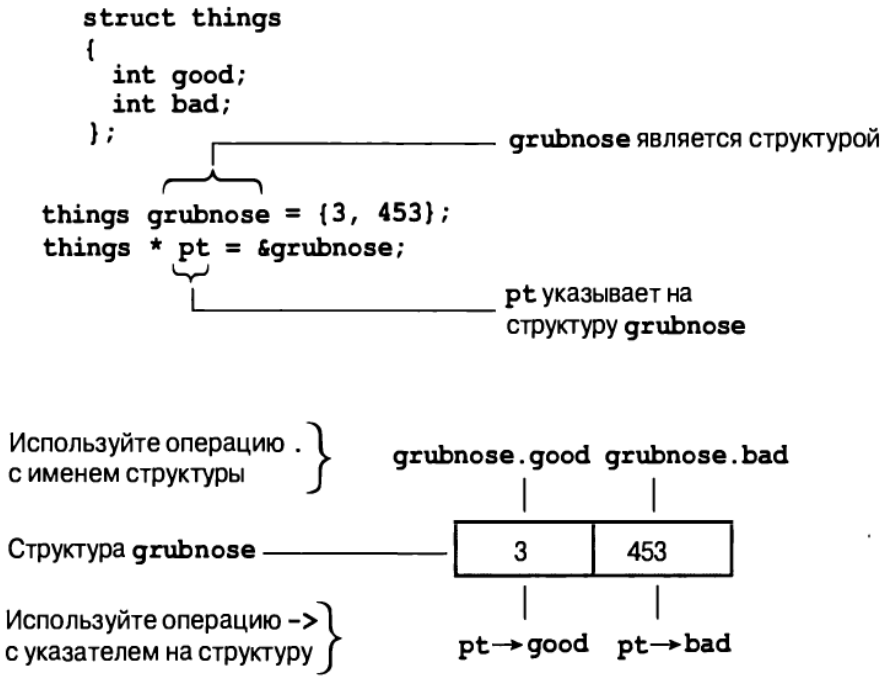


Рис. 3. Идентификация членов структуры.

Ещё один способ обращения к членам структур учитывает, что если `p_addr` — указатель на структуру, то `*p_addr` — сама структура. Поэтому, если `p_addr` — структура, то `(*p_addr).price` — ее член `price`. Правила приоритетов операций C++ требуют применения скобок в этой конструкции.

В следующем примере используется операция `new` для создания структуры, а также демонстрируются оба варианта нотации для доступа к ее членам.

Пример.

```
// newstrct.cpp - использование new со структурой
#include <iostream>
struct inflatable // определение структуры
{
   char name[20];
   float volume;
   double price;
};
int main ()
{
   using namespace std;
   inflatable *ps = new inflatable; // выделение памяти для структуры
   // ввод имени элемента inflatable
   cout << "Enter name of inflatable item ";
   cin.get(ps->name, 20);    // первый метод для доступа к членам
   // ввод объема в кубических футах
   cout << "Enter volume in cubic feet: ";
   cin >> (*ps).volume;      // второй метод для доступа к членам
   cout << "Enter price: $"; // ввод цены
   cin >> ps->price;
   cout << "Name: " << (*ps).name << endl;              // второй метод
   cout << "Volume: " << ps->volume << " cubic feet\n"; // первый метод
   cout << "Price: $" << ps->price << endl;             // первый метод
   delete ps;        // освобождение памяти, использованной структурой
   return 0;
}
```

Результаты работы программы (полужирным выделен ввод пользователя):

Enter name of inflatable item: **Fabulous Frodo**
Enter volume in cubic feet: **1.4**
Enter price: **$27.99**
Name: Fabulous Frodo
Volume: 1.4 cubic feet
Price: $27.99

④Указатели на структуры используются в двух ситуациях:

- для передачи структуры в функцию по указателю (не по ссылке!);
- для создания структур данных, основанных на динамическом распределении памяти (например, связанных списков).

Рассмотрим первое из этих применений.

Передача структур в качестве аргументов функции по значению имеет один существенный недостаток: в стек функции приходится копировать целую структуру. (*Вспоминаем, что аргументы, передаваемые по значению, копируются в стек вызовов функции целиком*.) Если структура невелика, дополнительные затраты памяти будут относительно небольшими. Однако, если структура состоит из большого количества членов или ее членами являются большие массивы, затраты ресурсов могут оказаться чрезмерными. Этого можно избежать, если передавать функции не сами структуры, а лишь указатели на них.

Если функции передается указатель на структуру (не ссылку!!!), в стек вызовов функции помещается только её адрес. В результате вызовы функции выполняются намного *быстрее*. Второе преимущество, которое проявляется в некоторых случаях, заключается в том, что, получив адрес, функция может *модифицировать* структуру, являющуюся ее фактическим параметром.

Пример. Рассмотрим программу, которая выводит на экран свою длительность работы в часах, минутах и секундах. Программа демонстрирует ранее не рассматриваемый способ передачи параметров в функцию по указателю!

```
1  #include <iostream>
2  #include <iomanip>
3  #include <conio.h> // kbhit()
4  #include <time.h>  // clock()
5  using namespace std;
6
7  struct my_time { // структура для хранения времени
8    int hours;
9    int minutes;
10   int seconds;
11 };
12
13 void display(const my_time *);
14 void update(my_time*);
15 void delay(int);
16
17 int main() {
18    my_time systime // для хранения длительности работы программы
19            {hours:0, minutes:0, seconds:0};
20    cout << "Длительность работы программы:\n";
21    for ( ; !kbhit(); ) { // пока не нажата клавиша
22       update(&systime);  // изменить длительность работы
23       display(&systime); // отобразить длительность
24    }
25    return 0;
26 }
27
28 void update(my_time *t) { // передача параметра по указателю
29    t->seconds++;  // изменение длительности работы программы
30    if(t->seconds==60) {
31       t->seconds = 0; t->minutes++;
32    }
33    if (t->minutes==60) {
34       t->minutes = 0; t->hours++;
35    }
36    if(t->hours==24) t->hours = 0;
37    delay(1000);  // задержка перед выводом на экран в 1 сек.
38 }
39
40 void display(const my_time *t) { // передача параметра по указателю
41    // выводим отметку времени
42    cout << setw(2) << t->hours   << "ч "
43         << setw(2) << t->minutes << "м "
44         << setw(2) << t->seconds << "с " << endl;
45 }
46
47 void delay(int d) {
48    int start_time=clock();  // сохраняем время от запуска программы
49    for (; clock()-start_time<d; ); //организация задержки в d миллисекунд
50 }
```

Пояснения к примеру:

Для хранения текущего времени от начала запуска программы объявляется структура `my_time` (см. строки программы 7-11), но не объявлена переменная этого типа. В основной программе объявляется и инициализируется значением 00:00:00 переменная `systime` типа `my_time` (строки 18-19). Это значит, что структура `systime` доступна только внутри функции `main()`.

Алгоритм работы основной программы (стр. 21-24) заключается в последовательном изменении длительности работы программы (функция `update()`) и выводе на экран очередной отметки

времени её выполнения (функция `display()`) с задержкой в 1 секунду (функция `delay()`). Указанные действия производятся до тех пор, пока пользователь не прервёт их нажатием на любую клавишу (условие завершения for стр. 21).

Изменение длительности работы программы. Функция `update()` принимает в качестве формального параметра указатель на структуру типа `my_time`. Признаком передачи указателя является символ `*` в её заголовке:

```
void update(my_time *t);
```

При этом в стек вызовов данной функции будет помещаться только адрес структурной переменной, а не целиком сама структура.

Поскольку локальная переменная `t` является указателем на структуру времени, то:

1) доступ к её полям возможен посредством оператора `->` :

```
t->hours
t->minutes
t->seconds
```

2) любые изменения, проводимые с полями структуры времени (например, `t->seconds++`), отражаются на значении фактического параметра `systime` в вызывающей программе. В частности для сброса таймера в 0 по достижении момента времени 24:00:00, в функции update() предусмотрен оператор

```
if (t->hours==24) t->hours = 0;
```

Этот оператор вынуждает компилятор извлечь адрес, хранящийся в указателе `t`, ссылающемся на структуру `systime` в функции `main()`, и присвоить члену `hours` значение 0.

После увеличения длительности работы программы, организована задержка её выполнении на 1 секунду с помощью библиотечной функции `clock()`, которая возвращает время, пройденное с момента запуска главной программы (см. функцию `delay()`).

Вывод на экран текущей длительности работы программы осуществляет функция `display()`. В качестве фактического параметра ей передаётся структурная переменная `systime` с помощью указателя не неё:

```
void display(const my_time *);
```

Квалификатор `const` в заголовке «говорит» о том, что внутри самой функции, изменений в передаваемом параметре не произойдёт.

Работа `display()` заключается в выводе новой отметки времени (стр. 42-44) в новой строке.

Обе функции `display` и `update` получают в качестве формальных параметров указатели на структуры. Следовательно, при их вызове необходимо в качестве фактических параметров задать адреса передаваемой структуры для обработки:

```
update(&systime);
display(&systime);
```

Указание адресов осуществляется посредством оператора косвенной адресации `&` на статическую структурную переменную `systime`.

Основные моменты в примере:

- Помните, что оператор "." используется для непосредственного обращения к членам структуры, а оператор "->" — для доступа к членам структуры через указатель на нее.
- В функциях `update()` и `display()` доступ к каждому члену структуры `systime` осуществляется через указатель.

Вопрос для самостоятельного изучения:

Учитывая, что `t` – указатель на структуру типа `my_time` в функции `display`, предложите другой способ доступа к значениям секунд, минут и часов.

Упражнения по программированию.

1. Измените программу вывода длительности её работы (предыдущий пример) так, чтобы значение структурной переменной `systime` в функции `display()` и `update()` передавалось по ссылке, а не по указателю, т.е. заголовки функции должны быть следующими:

```
void display(const my_time &);
void update(my_time &);
```

2. (Повторение пройденного в 1 семестре.) Измените функцию `display()` так, чтобы вывод отметки времени выполнялся всё время в одной строке и пользователь видел в любой момент времени всего две строки:

```
Длительность работы программы:
 0ч  4м 28с
```

Списки: линейные, односвязные, двусвязные

Второй случай использования указателей на структуры – это создание структур данных, основанных на динамическом распределении памяти, называемых связанными списками. Основными из них являются:

- линейные односвязные;
- линейные двусвязные;
- циклические односвязные;
- циклические двусвязные.

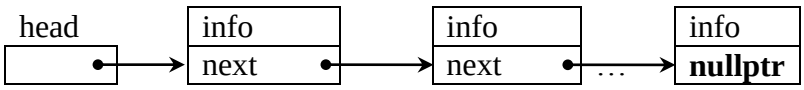
Линейный односвязный список.

На C++ реализуется с помощью структуры:

```
struct Elem {
   int info;  // информационное поле
   Elem * next; // указатель на следующий элемент
}
```

Информационное поле предназначено для хранения какой-либо информации может представлять собой структуру любой сложности. Другим обязательным полем является указатель на следующий элемент в списке, т.е. это будет указатель на такую же структуру.

Схематическое изображение линейного односвязного списка:



Последний элемент списка в качестве указателя на следующий элемент списка (которого нет) должен обязательно иметь значение нулевого указателя: `nullptr`.

Для работы со списком используется указатель на первый его элемент `head`:

```
Elem * head;
```

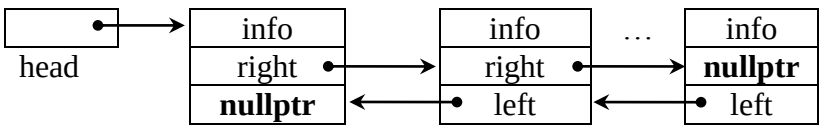
Линейный двусвязный список.

На C++ реализуется с помощью структуры:

```
struct Elem {
   int info;     // информационное поле
   Elem * left;  // указатель на элемент слева
   Elem * right; // указатель на элемент справа
}
```

Информационное поле также предназначено для хранения какой-либо информации и может представлять собой структуру любой сложности. Каждый узел линейного двусвязного списка содержит два указателя на два соседних элемента: «слева» и «справа».

Схематическое изображение линейного двусвязного списка:



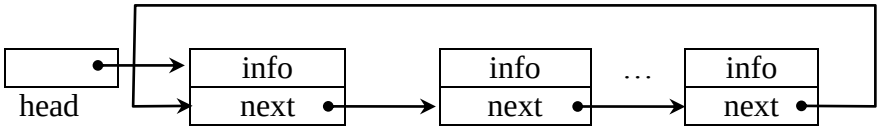
Крайние элементы списка в качестве указателей на отсутствующие элементы слева или справа должны иметь указатели `nullptr`. Для идентификации начала списка также используют указатель на его первый элемент `head`:

```
Elem * head;
```

Циклический односвязный список

Отличие от линейного односвязного списка заключается в том, что каждый узел списка ссылается на своего соседа. Реализуется на C++ с помощью такой же структуры `Elem` как и для линейного односвязного списка.

Схематическое изображение:



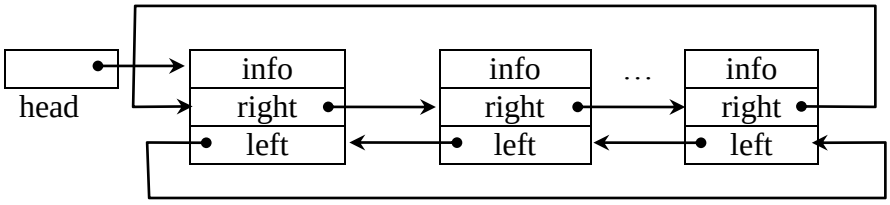
Обратите внимание, что узлов, которые никуда не указывают, нет. Для идентификации начала списка также используют указатель на его первый элемент `head`:

```
Elem * head;
```

Циклический двусвязный список

Отличие от линейного двусвязного списка заключается в том, что каждый узел списка ссылается на своего соседа слева и справа. Реализуется на C++ с помощью такой же структуры `Elem` как и для линейного двусвязного списка.

Схематическое изображение:



Для идентификации начала списка также используют указатель на его первый элемент `head`:

```
Elem * head;
```

Базовыми *операциями* по работе с динамическими списками являются вставка, удаление, поиск элемента, проверка списка на пустоту и вывод содержимого списка. Для знакомства с алгоритмами этих операций и получения дополнительной информации используем:

– п. 2 «Основные сведения» описания лабораторной работы №6 «Линейные связные списки».

– видео-материал «Связные списки» курса «Алгоритмы программирования и структуры данных» от университета ИТМО. Источник: https://openedu.ru