



Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

ОСНОВЫ

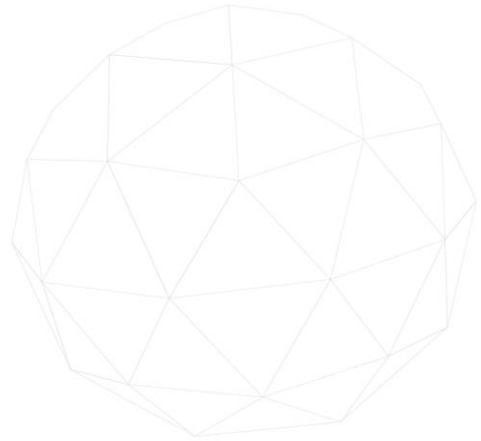
ПРОГРАММИРОВАНИЯ 8 ЧАСТЬ

ПЕРЕЧИСЛЕНИЯ

```
enum <имя-перечисления>  
    {<имя-константы 1> ,  
      < имя-константы 2>,  
      ...  
      < имя-константы N>;
```

Например:

```
enum color { r, g, b };
```



Слайд 200

Тема 8. Структуры, объединения, перечисления

Кроме базовых типов данных, рассмотренных в предыдущих темах, язык **C++** предоставляет простые и структурированные типы данных.

«Используя базовые, простые и структурированные типы данных, можно создавать производные типы данных, так называемые пользовательские типы данных.

Напомним, что к базовым типам относятся: целочисленный тип; вещественный тип; логический тип; символьный тип; строковый тип.

К структурированным типам относятся:

- массивы;
- структуры;
- файлы;
- объектный тип.»

Перечисления – средство создания типа данных посредством задания ограниченного множества значений. Значения данных перечисляемого типа указываются идентификаторами. Перечислением называется тип данных, который включает множество именованных целочисленных констант. Именованные константы, принадлежащие перечислению, называются

перечислимыми константам.

Для объявления перечислений используется ключевое слово **enum** .

Имя перечисления – это уникальный идентификатор, определяет тип перечисления, это пользовательский тип данных.

Имена констант являются элементами перечисления.

Объявление типа перечисления и переменной, которая имеет этот тип, может быть объединено в одну инструкцию. Пример объявления перечисления приведен на слайде:

color – имя типа перечисления, пользовательский тип;

r , g , b – сами перечислимые константы, это элементы перечисления.

ПЕРЕЧИСЛЕНИЯ

№	Объявление перечисления	Присвоенные значения
1	<code>enum color { r = 2, g = 2, b = 6};</code>	<code>r = 2, g = 2, b = 6</code>
2	<code>enum color { r = 2, g , b};</code>	<code>r = 2, g = 3, b = 4</code>
3	<code>enum color { r, g, b};</code>	<code>r = 0, g = 1, b = 2</code>
4	<code>color d;</code>	
5	<code>enum color { r, g, b } d ;</code>	
6	<code>color c = r;</code>	
7	<code>color c = 5;</code> Ошибка!	



Слайд 201

«Перечислимый тип описывает множество, состоящее из элементов-констант, иногда называемых нумераторами или именованными константами.

Значение каждого нумератора определяется как значение типа **int** , то есть целый. По умолчанию первый нумератор определяется значением ноль, второй - значением единица и так далее.»

Для инициализации значений нумератора не с нуля, а с другого целочисленного значения, следует присвоить это значение первому элементу списка значений перечислимого типа. Для перечислимого типа существует понятие диапазона значений, определяемого как диапазон целочисленных значений – первый пример на слайде.

При объявлении типа перечисления его значения могут инициализироваться произвольными целочисленными константами или константным выражением, как это показано в первом примере на слайде.

Если инициализация отсутствует, то перечислимым константам присваиваются последовательные значения, начиная от нуля, как показано во втором примере.

Если инициализированы не все элементы перечисления, то значения элементов, которые не определены, будут распределены соответственно, как показано в третьем примере

Переменная типа перечисление объявляется как показано в четвертом примере в таблице, например объявление переменная **d** .

Объявление типа перечисления и переменной, которая имеет этот тип, может быть объединено в одну инструкцию, как показано в пятом примере.

Переменным перечислимого типа можно присваивать только именованные значения перечислимых констант. Этот пример приведен в шестой строке в таблице. При необходимости можно явно задавать значения идентификатора, тогда очередные элементы списка получают последующие возрастающие значения.

В строке семь – ошибочное присвоение перечислимой константе.

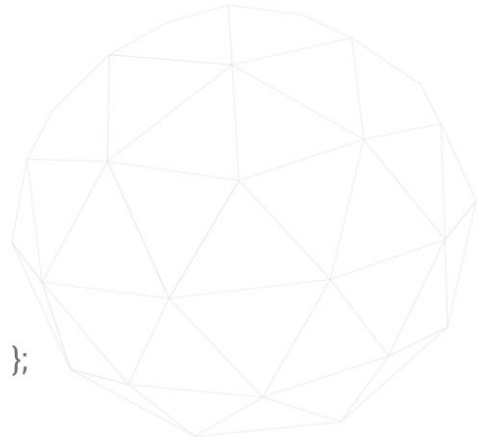
Целочисленным переменным можно присваивать только значения перечислимых констант.

СТРУКТУРЫ

```
struct <имя-структуры>
{
    <тип> <имя-элемент 1>;
    <тип> <имя-элемент 2>;
    ...
    <тип> <имя-элемент N>; };
```

Где

<имя-структуры> - пользовательский тип данных



Слайд 202

«В программировании есть много случаев, когда может понадобиться больше одной переменной для представления определенного объекта.

К счастью, язык **C++** позволяет программистам создавать свои собственные пользовательские типы данных — типы, которые группируют несколько отдельных переменных вместе. Одним из простейших пользовательских типов данных является структура. Структура позволяет сгруппировать переменные разных типов в единое целое.

Переменные типа структура подчиняются тем же правилам, что и обычные переменные. Следовательно, если вы хотите сделать переменную структуры доступной в нескольких файлах, то вы можете использовать ключевое слово **extern**».

Тип структура описывает упорядоченный набор данных, которые называются полями или элементами структуры. Каждое поле структуры имеет имя и тип, который должен отличаться от типов **void**. При этом следует учитывать, что структура может включать только такие поля, длина которых известна компилятору в момент определения структуры. Элементами структуры могут быть массивы и другие структуры.

При обработке большого объема прибегают к объединению информации в массивы. Недостаток массивов — это возможность объединения в массивы

только однотипной информации, одинакового типа.

Реально, создавая базу данных возникает необходимость в объединении информации разного типа.

Итак, структура – это объединение в единое целое множество поименованных элементов разного типа. Описание структуры выполняется с помощью зарезервированного слова **struct** . Это пользовательский тип данных, который строится на базе стандартных типов данных, допустимых на языке **C++** . Имя структуры - уникальный идентификатор. Имена структур принято писать с заглавной буквы, чтобы отличать их от имен переменных.

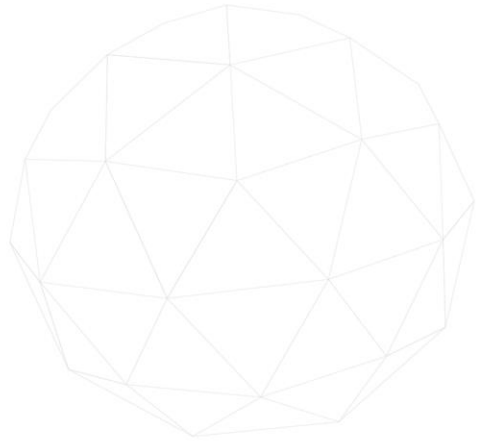
СТРУКТУРЫ

```
struct book
{
    char sur_name[10];
    char buk_name[20];
    int year;
    int pag; };
book buk, a[100], *pbuk;
```

Обращение к элементам структуры:

`buk.name [0]`

`a[i].year`



Слайд 203

Каждая структура включает в себя один или несколько объектов, называемых элементами структуры. Каждый элемент имеет тип и имя элемента. Описание каждого элемента заканчивается точкой с запятой. Элементы структуры заключаются в фигурные скобки, после которых также ставится точка с запятой. Отметим, что структуры даже одного типа нельзя сравнивать между собой.

Нельзя забывать ставить точку с запятой в конце объявления структуры. Это приведет к ошибке компиляции в следующей строке кода.

На слайде приведен пример структуры – база данных книголюба.

Имя структуры **buk** - пользовательский тип данных. Описанная структура состоит из четырех элементов:

- фамилия автора книги;
- название книги;
- год издания;
- количество страниц.
- Определены имя и тип каждого поля структуры.
- На слайде с помощью пользовательского типа, структуры, описаны:
- структура buk ;

- массив структур `a`;
- указатель на структуру `pbuk`.

С именем структуры не связан конкретный объект и с его помощью нельзя обращаться к элементам структуры. На слайде приведены примеры обращения к элементам структуры:

- первый пример – обращение к первому символу в фамилии автора;
- второй пример - год издания некоторой книги.

«Большим преимуществом использования структур, нежели отдельных переменных, является возможность передать всю структуру в функцию, которая должна работать с её членами. Это позволило нам не передавать каждую переменную по отдельности. Функция также может возвращать структуру. Это один из тех немногих случаев, когда функция может возвращать несколько переменных.»

ИНИЦИАЛИЗАЦИЯ ЭЛЕМЕНТОВ СТРУКТУРЫ

```
struct
{
    char sur_name[10];
    char buk_name[20];
    int year;
    int pag; } buk, a[100], *pbuk;
```

Инициализация элементов структуры:

```
buk={ "Маргарет Митчел", "Унесенные ветром", 1986, 586};
*pbuk=&buk;
```



Слайд 204

«Поскольку объявление структуры не провоцирует выделение памяти, то использовать предварительное объявление для нее вы не сможете. Если вы хотите использовать объявление структуры в нескольких файлах, чтобы иметь возможность создавать переменные этой структуры в нескольких файлах, поместите объявление структуры в заголовочный файл. Подключайте этот файл везде, где необходимо использовать структуру.»

Как и в случае с обычными переменными, элементы структуры не инициализируются автоматически и обычно содержат мусор. Инициализировать их нужно вручную.

Инициализация структур путем присваивания значений каждому члену по порядку — занятие довольно громоздкое, особенно, если этих членов много. В языке **C++** есть более быстрый способ инициализации структур - с помощью списка инициализаторов. Он позволяет инициализировать некоторые или все члены структуры во время объявления переменной типа **struct** :

Если в списке инициализаторов не будет одного или нескольких элементов, то им присвоятся значения по умолчанию, обычно это ноль.»

Если структура определяется один раз в программе, то при описании структуры можно опустить имя структуры, как это показано на слайде. После определения структуры, описаны переменные типа структура.

На слайде приведен пример инициализации элементов структуры.

Так как имя структурного типа обладает всеми правами типов переменных, то разрешено определять указатели на структуры.

Указатель может быть инициализирован. Значением указателя на структуру является адрес, а именно номер байта, начиная с которого структура размещается в памяти.

Указатель задает размер структуры и тем самым определяет, на какую величину, то есть на сколько байт, перемещается указатель на структуру, если к нему добавить или отнять единицу.

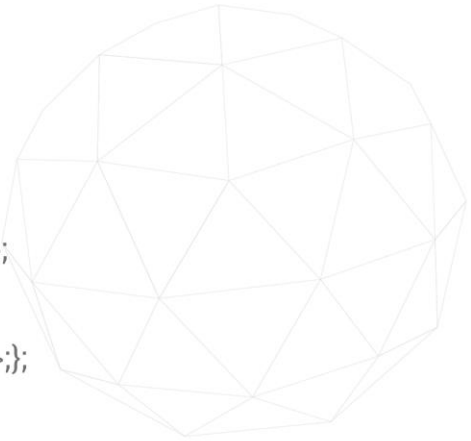
На слайде описан указатель на структуру с именем **pbuk** .

ОБЪЕДИНЕНИЯ

```
union <имя-объединения>
{
    <тип> <имя элемента1>;
    <тип> <имя элемента2>;
    ...
    <тип> <имя элементаN>;
};
```

Например:

```
union name
{
    int n;
    double f;
};
name d;
```



Слайд 205

«Объединение – это группирование переменных, которые разделяют одну и ту же область памяти. В зависимости от интерпретации осуществляется обращение к той или другой переменной объединения. Все переменные, что включены в объединение начинаются с одной границы.

Объединение позволяет представить в компактном виде данные, которые могут изменяться. Одни и те же данные могут быть представлены разными способами с помощью объединений.

Тип переменной, входящий в объединение, может быть:

- базовым типом;
- тип структура;
- тип объединения;
- тип класс.

Объединения — это структура данных, члены которой расположены по одному и тому же адресу. Поэтому размер объединения равен размеру его наибольшего члена. В любой момент времени объединение хранит значение только одного из членов.»

Итак, объединения - это объект, позволяющий нескольким переменным различных типов занимать один участок памяти. Объявление объединения

похоже на объявление структуры. Тип объединения описывает набор данных, которые называются элементами или членами объединения. Объединение позволяет хранить в одной и той же области памяти значения различных типов, которые соответствуют типам элементов объединения.

Как и в случае структуры, элементы объединения могут иметь любой тип за исключением типов `void` и функция.

Для объявления объединения используется ключевое слово **`union`** .

Пример объявления объединения приведен на слайде:

- `name` – имя объединения, это пользовательский тип, уникальный идентификатор;
- `n` , `f` – элементы объединения.

Каждый элемент объединения имеет имя и тип.

Описана переменная **`d`** , имеющая тип - объединение.

ОБЪЕДИНЕНИЯ

1. union name { int n; double f; } d;
2. name d = { 1 }; // d = 1
3. name d={1.0}; // Ошибка!
4. name b; b.n = 1; b.f = 1.1;



Слайд 206

«Использование объединения обозначает, что имеется намерение задействовать только одну ячейку памяти компьютера, в которую будут подставляться значения переменных, объединённых в одну группу и вероятно имеющих разные типы. Типы в **C++** ни что иное как подсказка компьютеру, сколько байтов памяти нужно отвести под переменную. Самый широкий тип, описанный внутри объединения, считается за подсказку, сколько памяти нужно выделить объединению. Объединения подобны структурам и пишутся объединения также как и структуры.

Длина объединения – это размер памяти в байтах, которая выделяется для одной переменной этого типа объединения.»

Объявление типа объединения и переменной, которая имеет этот тип, может быть объединено в одну инструкцию, как показано в первом примере.

При объявлении переменной типа объединения её можно инициализировать значением, которое должно иметь тип первого члена объединения. Во втором примере описана переменная **d** и инициализирована.

Третий пример – ошибочный, так как первый элемент объединения имеет целый тип.

Как и в случае со структурами, для доступа к элементу объединения используется точка, как показано в четвертом примере.

Объединения одного типа можно присваивать друг другу. В этом случае оператор присваивания выполняет по членное копирование объединений.

Также как и структуры объединения нельзя сравнивать.

Перейдем к задачам, позволяющим на примерах рассмотреть ваше изложенный материал.

МАССИВЫ СТРУКТУР

Создать массив структур, содержащий
информацию о товарах некоторого склада
О каждом товаре хранить:

- наименование
- количество
- цена единицы товара



Слайд 207

Элементы структуры называются полями структуры и могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него. Если отсутствует имя типа, должен быть указан список описателей переменных, указателей или массивов. В этом случае описание структуры служит определением элементов этого списка. Еще раз вспомним, что массив - это поименованный набор однотипной информации.

Создание структур, позволяет хранить в массиве информацию разного типа. На слайде приведена формулировка задачи, в которой создается массив, база данных, элементы которого – структура. Создается массив структур. Каждый элемент массива хранит информацию об одном наименовании товара.

Структура объединяет информацию разного типа: символьного типа, целого типа и вещественного типа.

Составим программу, которая позволит:

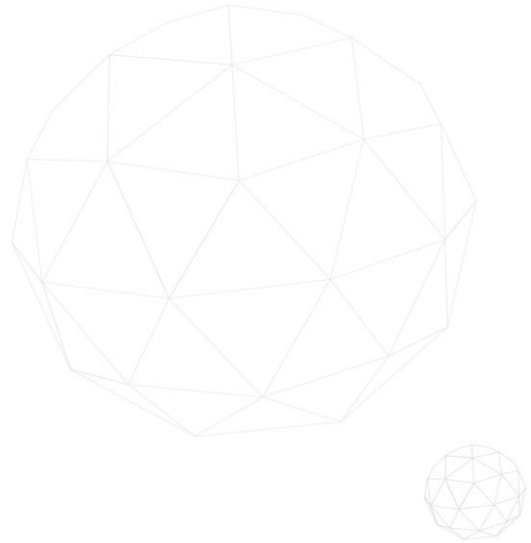
- создать массив структур, содержащий информацию о товарах некоторого склада;
- по введенному с клавиатуры наименованию, вывести информацию о соответствующем товаре, или вывести сообщение об отсутствии данного товара на складе;

- подсчитать общую стоимость товаров на складе.

Необходимо ввести переменную, определяющую количество наименований товаров и переменную для подсчета общей стоимости товаров.

ПРИМЕР 1. МАССИВ СТРУКТУР

```
#include<iostream>
#include<string.h>
#include<conio.h>
using namespace std;
int main()
{ struct Tovar
  { char name[30];
    int kol;
    float price; } array[100];
```



Слайд 208

На слайде представлен первый фрагмент программы.

Подключены заголовочные файлы:

- **iostream** - для организации ввода-вывода информации: заполнения базы данных, массива структур и вывода результатов вычислений;
- **string** – для сравнения наименований товаров;
- **conio** – для использования функции **getch** .

Определена структура – пользовательский тип с именем **Tovar** .

Описан массив структур. Размер каждого элемента массива определен описанной структурой:

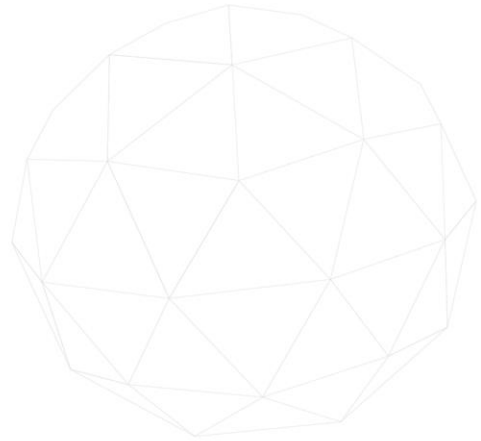
- переменная **name** занимает **30 байт**;
- переменная **kol** - **4 байта**;
- переменная **price** - **4 байта**.

Описанный массив структур состоит из тридцати элементов, где размер каждого элемента массива в байтах определяется размером структуры.

Еще раз напомним, что количество байт, отводимых под каждый тип информации, например, целый тип, зависит от компьютера, конкретной архитектуры компьютера.

ПРИМЕР 1. МАССИВ СТРУКТУР

```
int n; float s=0; char name[30];
cin>>n;
for( int i=0; i<n; i++)
{ cin>>array[i].name;
  cin>>array[i].kol;
  cin>>array[i].price;
}
```



Слайд 209

Для переменных одного и того же структурного типа определена операция присваивания, при этом происходит поэлементное копирование. Структуру можно передавать в функцию и возвращать в качестве значения функции. Другие операции со структурами могут быть определены пользователем. Размер структуры не обязательно равен сумме размеров ее элементов, поскольку они могут быть выровнены по границам слова.

Количество наименований товаров, количество элементов в массиве структур, определяет целая переменная **n**, значение которой вводится с клавиатуры.

Вещественная переменная **s** – введена для подсчета общей стоимости всех товаров.

Переменная **name** – наименование искомого товара, описан как символьный массив. Размер этой переменной, этого массива должен совпадать с размером элемента структуры, отведенного под хранение наименований товаров.

В нашем примере имя этой переменной совпадает с именем первого элемента структуры. Имена могут и не совпадать, а вот размеры – должны быть одинаковыми при описании, так как сравнение предполагается побайтно. Сравняться будут коды символов в наименованиях товаров из массива структур с кодами в наименовании искомого товара.

Совпадение имен не внесет разночтений, так как при обращении к элементам

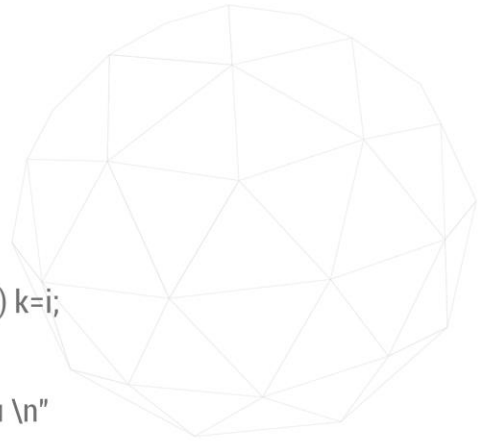
структуры присутствует имя массива и через точку указывается имя соответствующего элемента поля.

Рядом с именем массива в квадратных скобках указывается индекс, это фактически номер одного наименования товара в общем массиве структур.

В цикле вводится информация и создается массив структур. На каждой итерации цикла вводится информация об одном наименовании товара.

ПРИМЕР 1. МАССИВ СТРУКТУР

```
cout<<"\n Введите наименование";
cin>>name; int k=-1;
for( int i=0; i<n; i++)
{ if (strcmp(name, array[i].name)= =0) k=i;
  s+= array[i].kol * array[i].price; }
if (k== -1) cout<<"\n Товар не найден \n"
  else cout<<"\n "<<array[k].name<<array[k].price;
cout<<"\n Общая стоимость = "<<s;
getch(); }
```



Слайд 210

На слайде представлен последний фрагмент программы.

Переменная **k** – введена как признак наличия в массиве, базе данных, искомого товара, которой в цикле присваивается номер элемента, индекс массива с найденным наименованием товара. Первоначально этой переменной присваивается значение минус единица. Известно, что массив не может иметь отрицательного значения индекса.

В операторе цикла определена переменная **i** как параметр цикла.

Сравнивается наименование товара каждого элемента массива с наименованием искомого товара. Функция сравнения символьных массивов вырабатывает значение равно нулю, если прошло сравнение. Особенности этой функции мы рассматривали в предыдущей теме. Обращение к элементу структуры осуществляется через имя массива структур – **array** и имя конкретного поля структуры – **name**. Индекс **i** - определяет номер элемента в массиве структур.

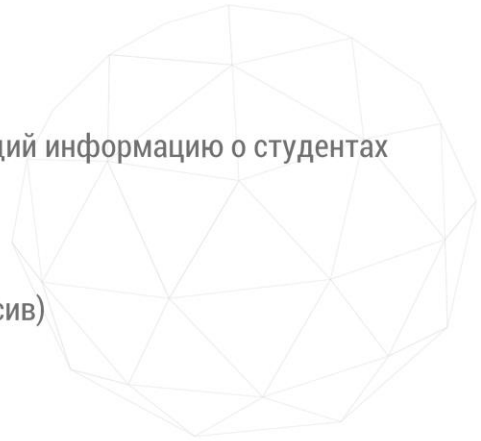
В случае, если наименование очередного товара совпало с наименованием искомого товара, переменной **k** присваивается номер этого элемента массива. Одновременно в этом же цикле подсчитывается общая стоимость товаров, информация о которых хранится в массиве структур. Для подсчета общей стоимости товаров определена переменная **s**.

По завершении цикла анализируется значение переменной **k** и выдается информация о данном товаре, по сохраненному номеру. Если значение этой переменной равно минус единице, значению, присвоенному в начале программы, выдается сообщение об отсутствии такого товара.

ПРИМЕР 2. МАССИВ СТРУКТУР

Создать массив структур, содержащий информацию о студентах

- фамилия
- год рождения
- оценки по трем дисциплинам (массив)



Слайд 211

Рассмотрим еще один пример обработки массива структур. На слайде определена структура массива, содержащая информацию о студентах некоторой группы.

Необходимо отсортировать массив структур. При сортировке необходимо выбрать ключ, по которому производится сортировка. Таким ключем в данной задаче является фамилия студентов. Будут сравниваться коды первой буквы в фамилиях. Напомним, что коды символов указаны в международной таблице **ASCII** кодов. Список отсортировать в алфавитном порядке по фамилиям студентов.

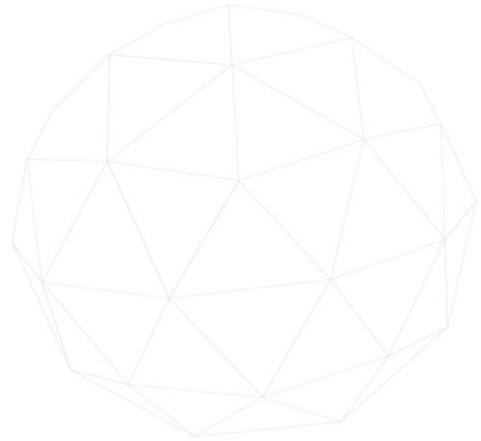
Каждая структура состоит из трех элементов. Первое поле структуры - массив символов, отведенный под хранение фамилий студентов.

Второе поле структуры содержит информацию целого типа - дата рождения.

Третий элемент структуры является массив, содержащий результаты сдачи экзаменов студентами по трем дисциплинам. Размер массива зависит от количества сданных студентами экзаменов. Размер всей структуры определяется входящими в нее элементами, количеством студентов в группе.

ПРИМЕР 2. МАССИВ СТРУКТУР

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{ struct Student
  { char name[20];
    int year;
    int bal[3]; };
```



Слайд 212

На слайде представлен первый фрагмент программы. Подключены заголовочные файлы.

Обычные массивы хранят однотипную информацию. Если необходимо хранить и обрабатывать информацию разного типа, прибегают к построению структур.

На слайде описаны элементы структуры:

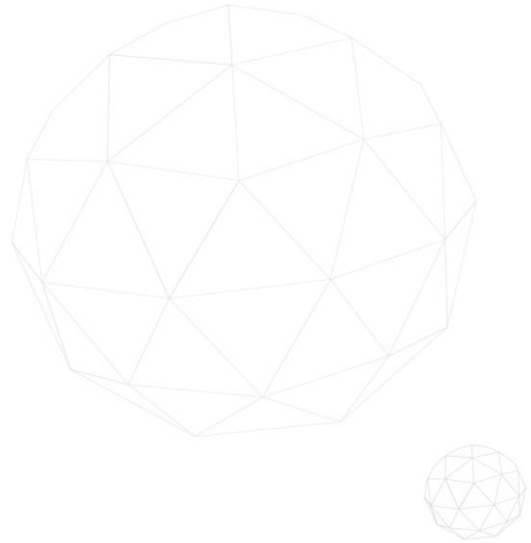
- фамилия студента, информация символьного типа. Под фамилию выделено 20 символов;
- дата рождения, целочисленный тип;
- оценки, полученные на экзаменах, массив целых чисел.

Элементами структуры могут быть объекты различных типов, массивы и даже структуры. Как известно, структура - это пользовательский тип данных, базирующийся на стандартных типах данных языка **C++**.

Описан пользовательский тип данных – структура с именем **Student**. Информация будет храниться в массиве структур.

ПРИМЕР 2. МАССИВ СТРУКТУР

```
int n; Student st_array[30], buf;  
cin>>n;  
// Ввод информации  
for( int i=0; i<n; i++)  
{ cin>>st_array[i].name;  
  cin>>st_array[i].year;  
  for( int j=0; j<3; j++)  
    cin>>st_array[i].bal[j]; }
```



Слайд 213

Рассмотрим следующий фрагмент программы. В данном фрагменте описан массив структур. Количество элементов в массиве зависит от количества студентов в группе.

В программе введены обозначения:

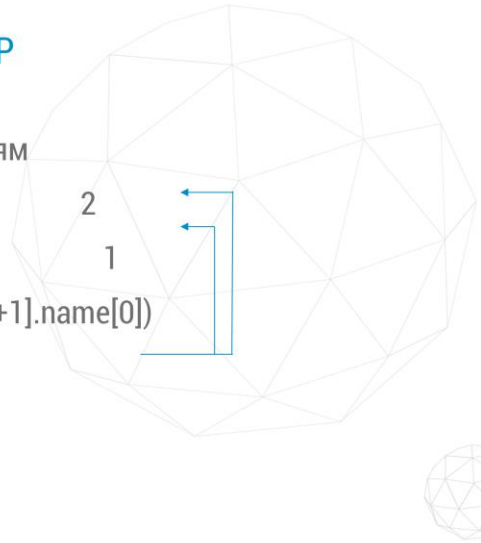
- **n** – количество студентов в группе;
- **buf** – переменная для обмена местами соседних элементов при сортировке, имеет тип структуры. Обмениваться будут не отдельные поля структур, а целиком вся структура;
- **st_array** - массив структур.

С клавиатуры вводится количество студентов, то есть значение переменной **n**. В цикле продемонстрирован ввод информации о студентах группы. На каждой итерации цикла вводится информация об одном студенте: фамилия, дата рождения и оценки по трем экзаменам. Для ввода оценок каждого студента организован еще один внутренний цикл.

Обратите внимание, при обращении к отдельным элементам структуры, указывается сначала имя структуры и через точку имя соответствующего элемента структуры.

СОРТИРОВКА МАССИВА СТРУКТУР

```
// Сортировка массива по фамилиям
for( int j=0; j<n; j++)
    for( int i=0; i<n-1; i++)
        if (st_array[i].name[0]>st_array[i+1].name[0])
            { buf= st_array[i];
              st_array[i]= st_array[i+1];
              st_array[i+1]=buf; }
```



Слайд 214

Сортировка массива структур выполняется во вложенных циклах.

Введены обозначения:

1 – внутренний цикл;

2 – внешний цикл.

Внутренний цикл позволяет один раз пройти по массиву структур, сравнивая соседние элементы. При обращении к массиву структур используют индекс массива, переменную **i**. Это номер студента в списке группы. Индекс ноль, указанный в переменной **name**, означает, что сравниваются коды первых символов в фамилиях студентов. Напоминаем о наличии нулевого элемента в массиве.

Обмен соседних элементов массива выполняется через переменную **buf**, которая имеет размер структуры. Вся информация об одном студенте, это есть один элемент массива структур.

Обратите внимание на закон изменения параметра внутреннего цикла, переменную **i**. На слайде выделено и подчеркнуто условие прекращения цикла. Последний раз в сравнении участвует предпоследний и последний элементы массива структур.

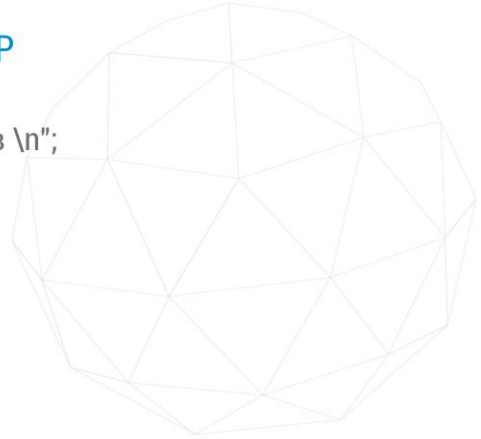
Внешний цикл продолжается до полной сортировки массива структур по

элементу - фамилия студента.

Функция **swap**, которую мы часто использовали для обмена переменных, не всегда адекватно работает со структурами. Поэтому используем переменную буфер для обмена соседних элементов структуры.

СОРТИРОВКА МАССИВА СТРУКТУР

```
cout<<"\n Отсортированный массив \n";  
for( int i=0; i<n; i++)  
{ cout<<"\n "<<st_array[i].name;  
  cout<<"\t "<<st_array[i].year;  
  for( int j=0; j<3; j++)  
    cout<<"\n "<<st_array[i].bal[j];  
  } getch(); }
```



Слайд 215

Последний фрагмент программы демонстрирует вывод на экран массива структур, отсортированного в алфавитном порядке по фамилиям студентов. На экран выводится соответствующее сообщение.

В программе используется структура вложенных циклов.

Во внешнем цикле выводится информация об одном студенте. Вывод экзаменационных оценок осуществляется во внутреннем цикле.

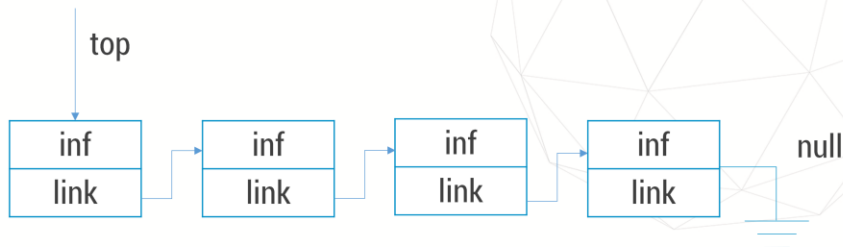
Для вывода оценок используется два параметра:

- параметр внешнего цикла - *i* , определяет номер студента в списке группы, в отсортированном массиве;
- параметр внутреннего цикла - *j* , перебирает все оценки, полученные данным студентом во время экзаменационной сессии.

Вывод организован так, что фамилия и дата рождения выводится на одной строке, а затем в столбик выводятся экзаменационные оценки. Управлять выводом можно с помощью управляющих символов перевода курсора на новую строку и управляющим символом табуляция.

ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

Объединение информации в динамические структуры:



top - указатель на начало списка

null - признак конца списка

Слайд 216

Мы уже рассмотрели такие понятия как: переменные-указатели; динамические массивы; рекурсивные функции; структуры; массивы структур.

Ознакомившись с перечисленными темами, рассмотрим динамические структуры данных, которые базируются на перечисленных понятиях.

«Структуры одного типа можно объединять не только в массивы. Их можно связывать между собой, создавая так называемые динамические структуры данных. Связь между отдельными структурами может быть организована по-разному, и именно поэтому среди динамических данных выделяют списки, стеки, очереди, деревья, графы. Для динамических данных память выделяется и освобождается в процессе выполнения программы, а не в момент ее запуска.

Список – абстрактный тип данных, реализующий упорядоченный набор значений. Списки отличаются от массивов тем, что доступ к их элементам осуществляется последовательно, в то время как массивы – структура данных произвольного доступа.

Список – это структура данных, представляющая собой конечное множество упорядоченных элементов, связанных друг с другом посредством указателей. Каждый элемент структуры содержит поле с какой-либо информацией, а также указатель на следующий элемент. В отличие от массива, к элементам списка нет произвольного доступа.

Так, например, если в программе объявлен массив из 100 элементов, то при запуске программы резервируется память для всех ста элементов, даже если в процессе работы программы всего будут использованы первые 10 элементов массива. С другой стороны, при использовании в программе динамических типов память под них заранее не выделяется. Лишь когда поступают новые данные, вызывается специальная функция, которая выделяет память, куда эти данные записываются.»

Схема связей динамических структур представлена на этом слайде.

ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

Динамическая структура данных характеризуется тем что:

- память выделяется в процессе выполнения программы;
- количество элементов может не фиксироваться;
- размерность структуры может меняться в процессе выполнения программы;
- может меняться характер взаимосвязи между элементами структуры



Слайд 217

«Динамические структуры данных в процессе существования в памяти могут изменять не только число составляющих их элементов, но и характер связей между элементами. При этом не учитывается изменение содержимого самих элементов данных. Такая особенность динамических структур, как непостоянство их размера и характера отношений между элементами, приводит к тому, что на этапе создания машинного кода программа-компилятор не может выделить для всей структуры в целом участок памяти фиксированного размера, а также не может сопоставить с отдельными компонентами структуры конкретные адреса. Для решения проблемы адресации динамических структур данных используется метод, называемый динамическим распределением памяти, то есть память под отдельные элементы выделяется в момент, когда они начинают существовать в процессе выполнения программы, а не во время компиляции. Компилятор в этом случае выделяет фиксированный объем памяти для хранения адреса динамически размещаемого элемента, а не самого элемента.»

Каждой динамической структуре данных сопоставляется статическая переменная типа указатель, ее значение – адрес этого объекта, посредством которой осуществляется доступ к динамической структуре.

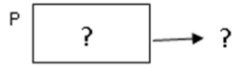


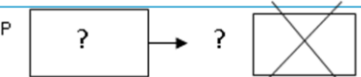
Сами динамические величины не требуют описания в программе, поскольку во время компиляции память под них не выделяется. Во время

компиляции память выделяется только под статические величины. Указатели – это статические величины, поэтому они требуют описания.

«Динамические структуры, по определению, характеризуются отсутствием физической смежности элементов структуры в памяти, непостоянством и непредсказуемостью размера, числа элементов структуры в процессе ее обработки.

Поскольку элементы динамической структуры располагаются по непредсказуемым адресам памяти, адрес элемента такой структуры не может быть вычислен из адреса начального или предыдущего элемента. Для установления связи между элементами динамической структуры используются указатели, через которые устанавливаются явные связи между элементами. Такое представление данных в памяти называется связным.»

ОПЕРАЦИИ С УКАЗАТЕЛЯМИ В ДИНАМИЧЕСКИХ СТРУКТУРАХ ДАННЫХ

Операция	Графическое изображение
1. <code>int *P;</code>	
2. <code>P = new int;</code>	
3. <code>P = Null;</code>	
4. <code>delete P;</code>	



Слайд 218

«Допустим, необходимо создать программу, позволяющую вводить данные о сотрудниках организации. Количество сотрудников неизвестно. Можно было бы создать массив записей с запасом. Однако, если данных о каждом сотруднике много, то каждая запись занимает много памяти. Получается, что мы будем расходовать много памяти в пустую. Проблема решается путем построения цепочки взаимосвязанных структур.»

Каждая динамическая структура характеризуется:

взаимосвязью элементов;

набором типовых операций над этой структурой.

Для организации связей между элементами динамической структуры, каждый элемент структуры должен содержать информационную часть и указатель, ссылку **link** на следующий элемент. В динамические структуры можно объединять любую информацию. Особенность такой организации информации заключается в том, что память под хранение информации может выделяться и освобождаться непосредственно во время выполнения программы.

На слайде представлены операции с указателями при создании динамических структур данных.

Первая операция: описание переменной - указатель находится в

неопределенном состоянии, после его описания.

Вторая операция - выделение памяти. Используется – оператор **new** . Данный оператор отводит место в оперативной памяти под хранение переменной указанного типа, в данном случае целого типа, и этот адрес вносится в указатель **p** .

Третья операция - указатель на пустой адрес, указатель никуда. Используется как признак конца списка.

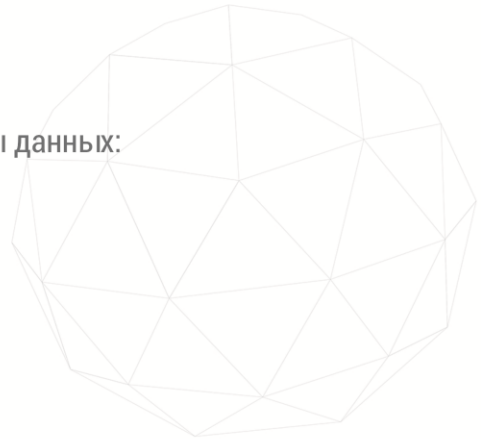
Четвертая операция - оператор **delete** освобождает память, занимаемую динамической переменной, и память становится доступна для других динамических переменных. Указатель опять находится в неопределенном состоянии.

ОДНОСВЯЗНЫЕ СПИСКИ

Описание динамической структуры данных:

```
struct <имя структуры>
{
    <тип> <переменная 1>;
    <тип> <переменная 2>;
    ...
    <тип> <переменная N>;
    <имя структуры> *<указатель>;};
```

Где, *<указатель> - указатель на следующий элемент



Слайд 219

Динамические структуры – это объединение информации в связанные динамические структуры:

- односвязные списки;
- двусвязные списки;
- бинарные деревья.

«Каждая компонента любой динамической структуры представляет собой структуру, содержащую, по крайней мере, два элемента: один - типа указатель, а второй – для размещения данных. В общем случае может содержать несколько элементов данных. Поле данных может быть переменной, массивом или структурой.»

Рассмотрим односвязные списки. На слайде приведено описание односвязного списка. Каждый элемент списка – это структура, содержащая информационную часть и указатель на следующий элемент структуры.

Каждый элемент односвязного списка имеет указатель, ссылку на следующий элемент списка. Тип указателя определяется через имя структуры. Имя структуры и есть пользовательский тип, базирующийся на стандартных типах данных языка **C++**.

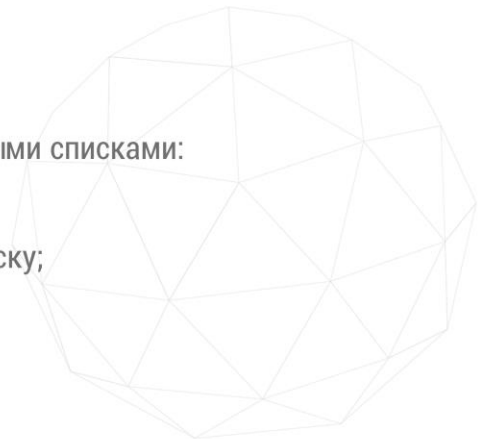
Односвязные списки можно формировать в виде:

- очереди, когда новый элемент добавляется в конец очереди, а удаляется элемент из начала очереди;
- стека, когда добавлять и удалять элемент можно только с одного конца списка, который называется вершиной стека.

ОДНОСВЯЗНЫЕ СПИСКИ

Основные операции над односвязными списками:

- переход от элемента к элементу;
- добавление нового элемента к списку;
- исключение элемента из списка;
- сортировка элементов



Слайд 220

«Основной задачей, при изучении динамических структур, является изучить особенности доступа к данным в динамических структурах, работу с памятью при использовании структур в программе.

В языке **C++** имеются средства создания динамических структур данных, которые позволяют во время выполнения программы образовывать объекты, выделять для них память, освобождать память, когда в них исчезает необходимость.

Динамические структуры данных — это любая структура данных, занимаемый объем памяти которой не является фиксированным. Размер подобной структуры ограничен только объемом оперативной памяти компьютера.

Если до начала работы с данными невозможно определить, сколько памяти потребуется для их хранения, память следует распределять во время выполнения программы по мере необходимости отдельными блоками. Блоки связываются друг с другом с помощью указателей. Такой способ организации данных называется динамической структурой данных, поскольку она размещается в динамической памяти и ее размер изменяется во время выполнения программы.

По сути своей это очень похоже на обыкновенный массив, с той лишь разницей, что размер его не имеет ограничений и содержит, объединяет информацию

различных типов.»

На слайде представлены основные операции над односвязными списками. Выполнение этих операций в массивах, процесс довольно трудоемкий.

Например, для удаления элемента массива, придется перетаскивать элементы массива на место удаляемого. В односвязном списке эта процедура выполнится одной операцией, изменением ссылки предыдущего элемента на последующий. Где текущий – это элемент, который надо удалить. Для добавления элемента в середину списка достаточно только перезамкнуть ссылки. Процесс сортировки можно реализовать непосредственно во время формирования динамической структуры.

ОБРАЩЕНИЕ К ЭЛЕМЕНТАМ ОДНОСВЯЗНОГО СПИСКА

Связь указателя с объектом:



Слайд 221

«Элемент динамической структуры в каждый момент может либо существовать, либо отсутствовать в памяти, поэтому его называют динамическим. Поскольку элементами динамической структуры являются динамические переменные, то единственным средством доступа к динамическим структурам и их элементам является указатель, адрес на место их текущего расположения в памяти. Таким образом, доступ к динамическим данным выполняется специальным образом с помощью указателей.»

Указатель содержит адрес определенного объекта в динамической памяти. Адрес формируется из двух слов: адрес сегмента и смещение. Сам указатель является статическим объектом и расположен в сегменте данных.

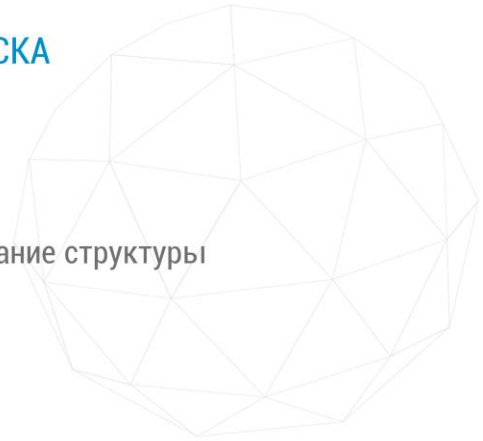
«Для обращения к динамической структуре достаточно хранить в памяти адрес первого элемента структуры. Поскольку каждый элемент динамической структуры хранит адрес следующего за ним элемента, можно, двигаясь от начального элемента по адресам, получить доступ к любому элементу данной структуры.»

Теперь перейдем к конкретным задачам и рассмотрим создание динамических структур данных.

ОПИСАНИЕ ОДНОСВЯЗНОГО СПИСКА

```
struct Produkt {  
    char name[10];  
    int kol;  
    Produkt *next; };  
#include<iostream>  
#include<conio.h>  
using namespace std;
```

описание структуры



Слайд 222

Решим задачу, на примере которой подробно рассмотрим операции над односвязными списками. Каждый элемент списка должен содержать информацию о наименовании товара, количестве данного наименования товара и ссылку на следующий элемент списка. То есть каждый элемент списка представляет собой структуру.

Требуется составить программу, формирующую односвязный список в виде стека о товарах склада, и распечатывает созданный список товаров.

Ввод прекращается, если вместо количества товаров ввести 0.

При создании односвязного списка в виде стека необходимо определить два указателя: на вершину стека и на текущий элемент. Указатель на вершину стека – фактически единственная связь со списком. Значение этого указателя нельзя терять, иначе потеряется связь со списком. Текущий указатель позволит перемещаться по списку от элемента к элементу.

На слайде представлен первый фрагмент программы. Описана структура с именем **Tovar**, состоящая из трех полей. Первые два поля – информационные, третье поле – указатель на следующий элемент динамической структуры, то есть указатель на следующий товар. Тип указателя должен совпадать с типом информации, на которую он указывает. Где **Tovar** – пользовательский тип данных.

Обратите внимание, структура описана вне всяких функций и даже до команд препроцессора. Это вполне допустимо.

РЕКУРСИВНАЯ ФУНКЦИЯ ОБХОДА ОДНОСВЯЗНОГО СПИСКА

```
// Рекурсивная функция печати элементов односвязного списка
void output( struct Produkt *p)
{ if (p==NULL) { cout<<"\n Конец списка \n";
                return; }
  cout<<"\n "<<p->name<<"\t"<<p->kol;
  output( p->next ); //обращение функции к самой себе
}
```



Слайд 223

В программе на слайде создана подпрограмма обхода и вывода на печать элементов односвязного списка, а именно наименование и количество товара.

Обход элементов односвязного списка можно реализовать в цикле. В данной задаче используется рекурсивная функция обхода и вывода на печать информации. Рекурсивной функции с именем **output** , представлена на слайде.

Данная рекурсивная функция не возвращает никакого результата, поэтому тип функции описан с помощью служебного слова **void** . В качестве формального параметра – указатель на начало односвязного списка. Тип указателя – это тип описанной выше структуры.

Функция рекурсивна обращается сама к себе, пока не будет пройден весь список. При каждом рекурсивном обращении – фактическим параметром является указатель на следующий элемент структуры.

Выход из процедуры осуществляется по достижению конца списка, то есть у последнего элемента структуры в поле указатель ставится константа **NULL** .
Формирование

«Доступ к данным в динамических структурах осуществляется с помощью операции стрелка, которую называют операцией косвенного выбора элемента структурного объекта, адресуемого указателем. Она обеспечивает доступ к элементу структуры через адресующий ее указатель того

же структурного типа.»

Обратите внимание, доступ к элементам структуры **name** и **kol**, осуществляется через указатель **p** и операцию косвенного выбора элемента структуры.

СОЗДАНИЕ ОДНОСВЯЗНОГО СПИСКА

```
int main()
{
    Produkt *top= NULL,*p;
    do { p = new tovar;
        cin>>p->name>>p->kol;
        if (p->kol != 0) {p->next=top; top=p;}
    } while(p->kol!=0);
    output(top); // обход и печать односвязного списка
    getch(); }
```



Слайд 224

На слайде представлена главная функция. Описаны два указателя:

Тор – на вершину, или начало списка;

Р – на текущий элемент.

Первоначально, указатель на начало списка устанавливается в неопределенное положение, константа **NULL**.

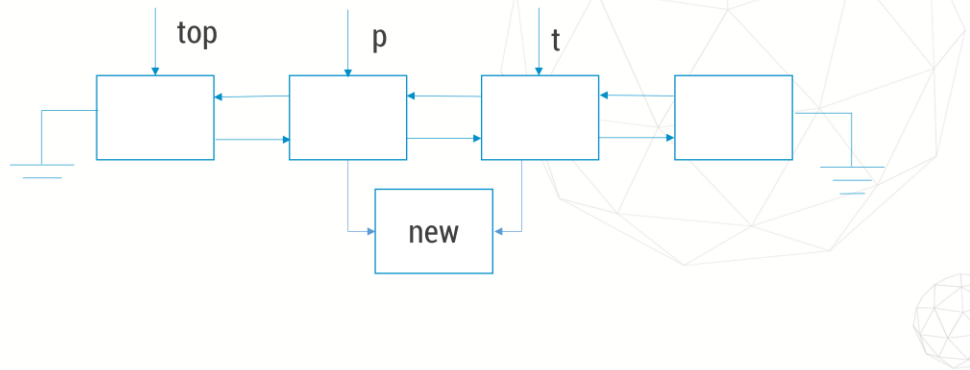
Организован цикл с постусловием для создания односвязного списка. Для каждого элемента односвязного списка с помощью оператора **new** выделяется динамическая память и адрес вносится в указатель. Информация вносится с клавиатуры и как бы привязывается к односвязному списку. Ввод продолжается до тех пор, пока в место количества товара, будет введен ноль.

После завершения ввода идет обращение к рекурсивной функции **output**, которая распечатает информацию из созданного односвязного списка.

«Имея возможность явного манипулирования с указателями, которые могут располагаться как вне структуры, так и внутри отдельных ее элементов, можно создавать в памяти различные структуры.

Однако необходимо помнить, что работа с динамическими данными замедляет выполнение программы, поскольку доступ к величине происходит в два шага: сначала ищется указатель, затем по нему – величина.»

ДВУСВЯЗНЫЕ СПИСКИ



Слайд 225

Элемент двусвязного списка представляет собой структуру, имеющую информационную часть и два указателя.

«Для ускорения многих операций целесообразно применять переходы между элементами списка в обоих направлениях. В таком списке каждый элемент, кроме первого и последнего, связан с предыдущим и следующим за ним элементами.

Двусвязный список более гибкий. При включении элемента в список, нужно использовать указатель как на элемент, за которым происходит включение, так и указатель на элемент, перед которым происходит включение. При исключении элемента из списка нужно использовать как указатель на сам исключаемый элемент, так и указатели на предшествующий или следующий за исключаемым элементы. Но так как элемент двусвязного списка имеет два указателя, то при выполнении операций включения или исключения элемента надо изменять больше связей, чем в односвязном списке.

Особое внимание следует обратить на то, что в отличие от односвязного списка здесь нет необходимости обеспечивать позиционирование какого-либо указателя именно на первый элемент списка, так как благодаря двум указателям в элементах можно получить доступ к любому элементу списка из любого другого элемента, осуществляя переходы в прямом или обратном направлении.

Однако по правилам хорошего тона программирования указатель желательно ставить на заголовок списка.»

Перечисленные процедуры сводятся к переключению указателей между элементами структуры. Опирая указателя **p** , на предыдущий элемент списка, и указателем **t** на последующий элемент, можно удалять, вставлять новый узел всего лишь перезамкнув указатели.

ДВУСВЯЗНЫЕ СПИСКИ

Описание двусвязного списка:

```
struct <имя структуры>
```

```
{
```

```
    тип <имя переменной 1>;
```

```
    тип <имя переменной 2>;
```

```
    ...
```

```
    <Имя структуры> previous, subsequent;
```

```
};
```

информационная часть



Слайд 226

На слайде приведено описание двусвязного списка.

Имя структуры – это уникальный идентификатор. Каждый элемент структуры имеет информационную часть и две ссылки:

- **previous** – указатель на предыдущий элемент;
- **subsequent** – указатель на последующий элемент.

Наличие ссылок на следующее звено и на предыдущее позволяет двигаться по списку от каждого звена в любом направлении: от звена к концу списка или от звена к началу списка, поэтому такой список называют двунаправленным.

В информационную часть могут входить переменные, массивы и даже структуры. Тип этих объектов определяет размер всей структуры.

Основные операции, осуществляемые с двусвязными списками:

- создание списка;
- печать списка;
- вставка элемента в список;
- удаление элемента из списка;
- поиск элемента в списке;
- проверка пустоты списка;

- удаление списка.

Операции для двунаправленного списка реализуются абсолютно аналогично соответствующим функциям для однонаправленного списка.

Просматривать двунаправленный список можно в обоих направлениях.

Для того, чтобы создать список, нужно создать сначала первый элемент списка, а затем при помощи функции добавить к нему остальные элементы.

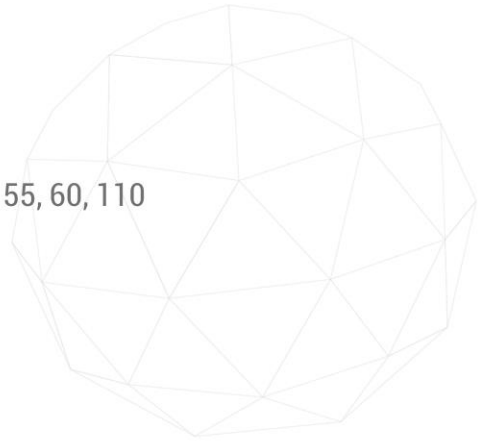
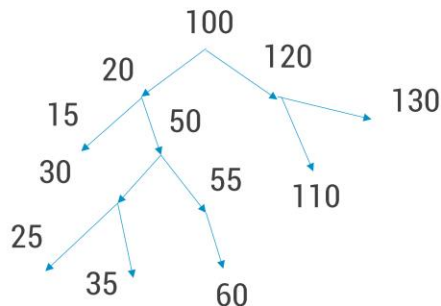
Добавление может выполняться как в начало списка, так и в конец списка.

БИНАРНЫЕ ДЕРЕВЬЯ

Последовательность чисел:

100, 20, 120, 130, 50, 15, 30, 35, 25, 55, 60, 110

top – вершина дерева



Слайд 227

«Бинарное дерево является одним из важнейших и интересных частных случаев графа. Древовидная модель оказывается довольно эффективной для представления динамических данных с целью быстрого поиска информации.

Дерево - это структура данных, представляющая собой совокупность элементов и отношений, образующих иерархическую структуру этих элементов. Каждый элемент дерева называется вершиной или узлом дерева. Вершины дерева соединены направленными дугами, которые называют ветвями дерева. Начальный узел дерева называют корнем дерева, ему соответствует нулевой уровень. Листьями дерева называют вершины, в которые входит одна ветвь и не выходит ни одной ветви.

Все вершины, в которые входят ветви, исходящие из одной общей вершины, называются потомками, а сама вершина – предком.»

Бинарное дерево - это такая динамическая структура, в которой:

- есть только один узел, в который не входит ни одного ребра, который называется вершиной дерева;
- в каждый узел, кроме вершины, входит только одно ребро;
- из каждого узла исходит не более двух ребер.
- При построении бинарного дерева используется следующее правило:

- от каждого узла влево-вниз - ставится меньший узел;
- от каждого узла вправо-вниз - ставится больший узел.

Преимуществом построения бинарных деревьев является быстрота поиска информации.

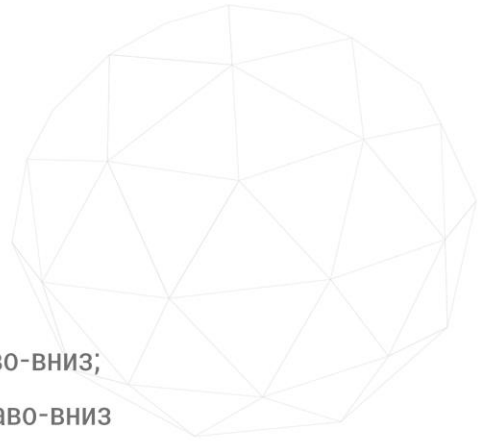
На слайде представлена последовательность чисел, с которыми построено бинарное дерево по перечисленным выше правилам. При описании бинарных деревьев каждый элемент структуры имеет два указателя: на элемент, стоящий вправо-вниз и на элемент, стоящий влево-вниз.

БИНАРНЫЕ ДЕРЕВЬЯ

```
struct Tovar {char name[10];  
             int kol;  
             Tovar *l,*r;  
             };
```

l - указатель на узел, стоящий влево-вниз;

r - указатель на узел, стоящий вправо-вниз



Слайд 228

Рассмотрим описание структуры - бинарное дерево. Рассмотрим основные принципы построения бинарных деревьев.

Каждый элемент структуры имеет информационную часть и два указателя: на элемент, расположенный влево вниз от данного узла и на элемент, расположенный вправо вниз от данного узла.

Основные операции над бинарными деревьями:

- переход от элемента к элементу;
- добавление нового элемента к списку;
- исключение элемента из списка;
- сортировка элементов.

При удалении элемента бинарного дерева руководствуются следующим правилом: надо от удаляемого узла сделать шаг влево и идти до конца вправо. Тогда бинарное дерево не рассыпится.

«Упорядоченное дерево – это дерево, у которого ветви, исходящие из каждой вершины, упорядочены по определенному критерию»

Деревья являются рекурсивными структурами, так как каждое поддерево также является деревом.

Действия с рекурсивными структурами удобнее всего описываются с помощью рекурсивных алгоритмов.»

Введение в языки программирования С и С++ | Уроки С++ - Ravesli /
<https://ravesli.com/urok-2-vvedenie-v-yazyki-programmirovaniya-c-i-s/>

Т, А. Павловская С/С++ Программирование на языке высокого
уровня/<http://cph.phys.spbu.ru/documents/First/books/7.pdf>

Введение в программирование | Уроки С++ - Ravesli/<https://ravesli.com/urok-1-vvedenie-v-programmirovanie/>

Технология программирования - Информатика, автоматизированные
информационные технологии и системы/
https://studref.com/441961/informatika/tehnologiya_programmirovaniya

Бьерн Страуструп. Язык программирования С++ 11/ https://vk.com/doc-145125017_450056359

Язык СИ++ Учебное пособие / <http://5fan.ru/wievjob.php?id=4301>