# Программирование на языке C++

Зариковская Наталья Вячеславовна

Лекция 4

# Составные типы

К составным типам данных можно отнести:

- перечисление;
- структуру;
- смесь.

Общим для этих типов является то, что они встраиваются в программу пользователя только на этапе компиляции.

При этом компилятору известны (определены) только правила их создания (описания).

Сами же эти типы создаются пользователем, согласно этим правилам, по его усмотрению, с учетом конкретных требований, решаемых задач. ***Таким образом, составные типы можно рассматривать как механизм расширения стандартных типов данных.***

# Перечислимый тип

- Перечислимый тип определяет упорядоченное множество значений путём перечисления идентификаторов, взятых в скобки и отделяемых друг от друга запятой.
- Этот тип можно рассматривать как способ задания  мнемонических констант, а так же получения дополнительной возможности контроля изменения значения вводимой переменной.
- Его определение имеет две формы:

- **enum{список идентификаторов };**
- и
- **enum enum_ind {список идентификаторов }; ,**
- 
- где enum -  зарезервированное слово,  используемое для объявления перечислимого типа; enum_ind - идентификатор, определяемого пользователем  «перечислимого» типа; « список идентификаторов» - список идентификаторов,  разделённых символом  запятая.

# Перечислимый тип

- Первая форма  определения используется сугубо для задания мнемонических констант и может использоваться только один раз в пределах видимости (блок, файл). Целью, в данном случае, служит возможность использования идентификаторов вместо числовых значений.
- Вторая форма определения используется для задания идентификатора нового типа, применяемого при описании переменных.  Описание переменных перечислимого типа выполняется двумя способами: обычным,  где сначала определяется тип - перечисление, а затем описывается  переменная (см. ‘ оператор объявления ’); одновременно с описанием перечислимого типа  объявляется переменная, по следующей форме -
- **enum enum_ind {список идентификаторов } идентификатор_переменной; .**

# Перечислимый тип

- Значения идентификаторов из списка идентификаторов могут задаваться либо по умолчанию, либо путём явной инициализации: по умолчанию - первому идентификатору присваивается значение ноль, а каждому последующему на единицу больше;
- путём явной инициализации может быть определены значения любых идентификаторов, а каждые последующие, относительно заданных, будут на единицу больше.

- **Например:**

- enum{a,b,c,d}; //a=0,b=1,c=2,d=3
- enum{a,b=7,c=67,d}; //a=0,b=7,c=67,d=68
-
- Следует обратить внимание, что способ явной инициализации допускает одинаковые значения мнемонических констант.

-

# Перечислимый тип

- Приведенные ниже две простейшие программы покрывают практически все возможности использования перечислимых типов.

```
#include <iostream.h>
void main( )
{ enum dd{a,b,c,d};  // эквивалентное определение переменной ‘i’ - enum dd{a,b,c,d}i;
 dd i; int f;                // f - технологическая переменная
for( i=a;i<=d;i++)
cout<<a<<b<<c<<d<<”\n”;
cin>>f;
}
#include <iostream.h>
void main()
{enum {a,b,c,d};
int f;
for(int i=a;i<=d;i++)
cout<<a<<b<<c<<d<<”\n”;
cin>>f;

}
```

# Структура

- Структура - упорядоченная совокупность произвольных типов данных, объединённых в одной области памяти. Тип структуры вводится описанием следующего вида
- **struct[ имя_struct]{тип 1 имя_поля 1;**
- **тип 2 имя_поля 2;**
- **……..**
- **тип n  имя_поля n;};**
- 
- где имя_struct - имя структуры шаблона, удовлетворяющего правилам задания идентификаторов языка C++; тип 1, тип 2, ..., тип n - любые предопределённые типы ;
- имя_поля 1, ... , имя_поля n - идентификаторы полей, удовлетворяющие правилам задания идентификаторов.
- Например:
- struct tovar {char name[10];
- char nazn[15];
- int time;
- int price;};
- 

# Структура

- Описание структуры представляет собой задание нового типа struct имя_struct и не приводит к выделению памяти, а лишь даёт информацию компилятору о типах и количестве полей. Эта информация используется компилятором при описании структурных переменных для резерви рования необходимого места в памяти и организации доступа к необходимым полям структурной переменной.
- Описание структурной переменной состоит из задания типа и имени структурной переменной, и имеет вид

- **struct[ имя_struct] имя_var_struct;**

# Структура

- В языке С допускается совмещение описания шаблона структуры и структурных переменных.

- **Например:**

*Struct tovar1 { char name[10]; long int price;}tov1, tov2;*

- где tov1,tov2 - наименование структурных переменных (tov1,tov2 - переменные типа struct tovar1).
- Доступ к полям структурных переменных производится с помощью оператора- «.», который формирует ссылку на нужное поле_i структурной переменной и имеет вид -
- имя_var_struct.поле_i.
- Такая ссылка может располагаться в любом месте, где допустимы ссылки на простые перемен ные.
- Например:
- tov1.name=’’ volga’’,
- tov2.price=12000;
-

# Структура

- Ссылка на поле структурной переменной обладает всеми свойствами обычных переменных. Например, если поле это массив символов (char name[10];), то tov1.name - указатель- константа на первый элемент этого массива, а выражение &tov1.price - есть взятие адреса первого байта поля price. Нет, также, отличий в правилах видимости и времени существования от обычных переменных.
- В том случае, когда в функции определена лишь одна структура, то допустимо использование описания без указания имени. Например:
- struct{ char name[10];
- long int price;} tov1;
- Переменные типа структура (элементы структуры) могут быть инициализированы соответствующими выражениями в вложенных фигурных скобок. Например:
- struct{ char name[10]; long int price;} tov1={"стол",100};
-

# Структура с битовыми полями

- При программировании задач низкого уровня часто неудобно пользоваться битовой арифметикой. В этом случае используются структура, элементом которой служит битовое поле, обеспечивающее доступ к отдельным битам памяти. Тип структуры с битовым полем вводится описанием следующего вида:
- **struct [ имя_struct]{ unsigned имя_поля 1: длина_поля 1;**
- **unsigned имя_поля 2: длина_поля 2;**
- **……..**
- **unsigned имя_поля n: длина-поля n ;};**
- длина_поля определяет число битов, отводимых соответствующему полю и задается целым выражением или константой. Длина поля может иметь нулевую длину, что обозначает выравнивание на границу следующего слова.
- Вне структур битовые поля объявлять нельзя. Нельзя также организовывать массивы битовых полей и нельзя применять к полям операцию определения адреса.
- Пример:
- **struct { unsigned key : 1;**
- **unsigned keycod : 3;**
- **unsigned keytime: 5;**
- **} mykey;**

# Структура с битовыми полями

- Битовые поля могут содержать и знаковые компоненты. Эти компоненты автоматически размещаются на соответствующих границах слов, при этом, некоторые биты слов могут оставаться неиспользованными.

- Ссылки на поле битов выполняются точно так же, как и компоненты общих структур.

- Например:
- **Mykey.keycod=3;**

- Общее представление  битового поля рассматривается как целое число, максимальное значение которого определяется длиной поля.

-

# Объединение (Смесь)

- Объединение это тип данных, определяемый пользователем, позволяющий организовать размещение в одной и той же области памяти объекты различных типов. Объединение в общем виде записывается:
- **union ind_un {тип1 идентификатор1;**
- **тип 2 идентификатор2;**
- **.....**
- **тип n идентификатор n;},**
- где ind_un - определяемый новый тип ‘’ объединение’’; тип1 - тип n - предопределённые на момент объявления типы переменных идентфикатор1 - идентификатор n. Следует отметить, что тип идентификатора (часто по аналогии со стуктурой называемый полем) может быть любым в том числе и структурой.
- Работа оператора объявления типа ‘’объединение’’ состоит в том, что компилятору поставляется информация о том, что для хранения переменных будет выделена одна и та же область памяти. В этом случае для переменной типа union выделяется места в памяти ровно столько, сколько необходимо для размещения в памяти элементу union имеющему наибольший размер в байтах. При этом необходимо помнить, что транслятор не знает, какой член используется в данный момент, и поэтому контроль типа невозможен.
- Одновременно в памяти может находиться значение и быть «активным» только один элемент объединения.

# Объединение (Смесь)

- Описание переменной union состоит из задания типа (union ind_un) и идентификатора (имя_var_ union) переменной, и имеет вид
- **union ind_un имя_var_ union;**
- Аналогично структуре в языке С допускается совмещение описания шаблона объединения и переменных объединения.
- Например:
- union ind_un {
- int i; //требуется 2 байта
- double j; //требуется 8 байт
- char k; //требуется 1 байт
- } m, *mptr=&m;
-
-

# Объединение (Смесь)

- Для хранения переменной m типа union ind_un будет выделено 8-байт (значения double).
- Доступ к элементам объединения выполняется при помощи селекторов элемента структуры (операция "." или
- "->"), например:
- m.i = 99; или
- mptr->i=99;
- В том случае, когда m содержит объект типа int или char, остаются неиспользованные байты (6 байт, 7 байт, туда помещаются символы-заполнители).
- Допускается инициализация переменных объединения через элемент, объявленный первым: Например:
- union ind_un {
- int i;
- double j;
- char k;
- } m={99};
-
-
-

# Объединение (Смесь)

- В последних версиях C++ допускаются безымянные объединения. Объединение, не имеющее идентификатора типа (тега) и не используемое для объявления поименованного объекта (или другого типа) называется безымянным объединением. Оно имеет следующий вид:
- **union { тип1 идентфикатор1;**
- **тип 2 идентфикатор2;**
- **.....**
- **тип n идентификатор n;},**
- 
- Доступ к его элементам осуществляется непосредственно в той области действия, в которой объявлено это объединение, без использования синтаксиса x.y или p->y.
- Безымянные объединения не могут иметь функции элементы и на файловом уровне должны быть объявлены как статические. Другими словами, безымянное объединение может не иметь внешних связей.
- 
- 
- 

# Переменные с изменяемой структурой

- При решении некоторых задач приходится иметь дело с объектами программы относящихся к одному и тому же классу и отличающихся между собой лишь некоторыми деталями. Например, при составлении базы данных "службы занятости" вместо задания множества структур можно обойтись одной с переменной структурой. Тип переменной структуры вводится описанием следующего вида
- **struct{// общие компоненты;**
- тип 1 имя_поля 1;
- тип 2 имя_поля 2;
- тип n имя_поля n;
- **метка активного компонента;**
- **union { // описание индивидуальных компонент ;**
- тип1 идентфикатор1;
- тип 2 идентфикатор2;
- .....
- тип n идентификатор n;} **идентификатор-объединения;**
- **} идентификатор-структуры;**
-
-

# Переменные с изменяемой структурой

- Как видно переменная структура состоят из трех частей: набора общих компонент, метки активного компонента, отвечающей за выбор переменной компоненты, и части с меняющимися компонентами. Такая структура называется переменной структурой, потому что часть её индивидуальный компонент меняются в зависимости от значения метки активного компонента.
- Рекомендуется в качестве активного компонента использовать перечисляемый тип, что позволяет избежать ошибок при выборе переменных структуры.
- Обращение к элементам изменяемой структуры выполняется обычным образом - при помощи составного имени.
- Итак, на настоящий момент мы с вами познакомились с основными встроенными типами C++ (т.н. простыми типами). Теперь мы вернемся к типам данных через несколько лекций, чтобы рассмотреть сложные типы (массивы, указатели, ссылки и т.п).
-
-
-
-

# Краткие выводы из содержания лекции

- 1) в C++ различаются прописные и строчные буквы
- 2) символьная константа  'x' - просто  другая запись для  целой константы 120.
- 3) строки в C++ заканчиваются нулевым символом
- 4) определение и объявление переменной - разные понятия
- 5) до использования переменная должна быть определена
- 6) определения могут быть в программе где угодно.
-
- ***Вы должны быть в состоянии полностью объяснить программу (раздатка)***

- 1) объявление структуры  задает только тип.  После этого можно описывать переменные такого типа.
- 2) для доступа к элементу структуры через саму эту структуру используется операция ".", а через указатель на нее - операция "->".
- 3)  структура  может  содержать  указатель  на  структуру  такого же типа ("ссылку на себя").
- 4) если имя типа получается слишком сложным, то можно описать его одним
- идентификатором с помощью typedef.
-