Тема 1.1 Базовые типы и агрегаты данных в языке «С»

В языке программирования С присутствуют следующие типы данных:

Тип	***Типичный размер в битах***	***Минимально допустимый диапозон значений***
char	8	от -127 до 127
unsigned char	8	от 0 до 255
signed char	8	от -127 до 127
int	16 или 32	от -32767 до 32767
unsigned int	16 или 32	от 0 до 65535
signed int	16 или 32	то же, что int
short int	16	от -32767 до 32767
unsigned short int	16	от 0 до 65535
signed short int	16	то же, что short int
long int	32	от -2 147 483 647 до 2 147 483 647
long long int	64	от $-(2^{63}-1)$ до $(2^{63}-1)$, добавлен стандартом С99
signed long int	32	то же, что long int
unsigned long int	32	от 0 до 4 294 967 295
unsigned long long int	64	от 0 до $(2^{64}-1)$, добавлен в С99
float	32	от 1E-37 до 1E+37, с точностью не менее 6 значащих десятичных цифр
double	64	от 1E-37 до 1E+37, с точностью не менее 10 значащих десятичных цифр
long double	80	от 1E-37 до 1E+37, с точностью не менее 10 значащих десятичных цифр

Перечислимый тип задаёт тип, который является подмножеством целого типа.

Объявление переменной перечислимого типа задаёт имя переменной и определяет список именованных констант, называемый списком перечисления:
enum [*<тег>*] {*<список перечисления>*} *<описатель>* [, *<описатель>* ...]; enum *<тег>* *<описатель>* [, *<описатель>* ...];

Тег предназначен для различения нескольких перечислимых типов, объявленных в одной программе.

Список перечисления содержит одну или более конструкций вида:
<идентификатор> [= *<константное выражение>*]

Конструкции в списке разделяются запятыми. Каждый *идентификатор* именует элемент списка перечисления. По умолчанию, если не задано *константное выражение*, первому элементу присваивается значение 0, следующему элементу – значение 1 и т.д.

Запись = <константное выражение> изменяет умалчиваемую последовательность значений. Элемент, идентификатор которого предшествует записи = <константное выражение>, принимает значение, задаваемое этим константным выражением. Константное выражение должно иметь тип int и может быть как положительным, так

и отрицательным. Следующий элемент списка получает значение, равное <константное выражение> + 1, если только его значение не задаётся явно другим константным выражением.

В списке перечисления могут содержаться элементы, которым сопоставлены одинаковые значения, однако каждый идентификатор в списке должен быть уникальным. Кроме того, идентификатор элемента списка перечисления должен быть отличным от идентификаторов элементов всех остальных списков перечислений, а также от других идентификаторов.

```
enum Weekdays {SA, SU, MO, TU, WE, TH, FR};
enum Weekdays {SA, SU = 0, MO, TU, WE, TH, FR};   // SA и SU имеют одинаковое значение
void main()
{ enum Weekdays d1 = SA, d2 = SU, d3 = WE, d4;
d4 = 2;                                           // Ошибка!
d4 = d1 + d2;                                     // Ошибка!
d4 = (enum Weekdays)(d1 + d2);                    // Можно, но результат
d4 = (enum Weekdays)(d1 - d2);                    // может не попасть
d4 = (enum Weekdays)(TH * FR);                    // в область определения
d4 = (enum Weekdays)(WE / TU);                    // перечисления
}
```

Объявление без последующего списка описателей описывает тег, или, если так можно сказать, шаблон перечисления.

Структура - это набор данных, где данные могут быть разного типа. Например, структура может содержать несколько переменных типа int и несколько переменных типа char. Переменные, которые содержаться в структуре называются членами или полями структуры. Структуры можно определять с помощью ключевого слова struct.

Пример описания структуры:

```
struct student
{
   char name[50];
   int kurs;
   int age;
};
```

Мы определили структуру в которую входят переменные kurs, age и массив name. В этом описании student является шаблоном структуры, struct student является типом данных. После описания структуры нужно ставить точку с запятой. Чтобы использовать структуру необходимо объявить переменные типа struct student.

Например,

```
struct student s1, s2;
```

Переменные s1 и s2 являются переменными типа struct student. Компилятор автоматически выделит память под эти переменные. Под каждую из переменных типа структуры выделяется непрерывный участок памяти.

Для получения доступа к полям структуры используется операция **точка**. Например,

```
strcpy(s1.name, "Бардин Павел");
s1.kurs=3;
s1.age=20;
```

В языке С есть возможность объявлять переменные структуры при описании структуры:

```
struct student
{
  char name[50];
  int kurs;
  int age;
} s1, s2;
```

Переменные s1 и s2 являются переменными типа struct student.

Элементами или полями структуры могут быть переменные, массивы, ранее определенные структуры. Функции не могут быть полями структуры (В языке Си). В языке С++ функции могут быть полями структуры и такие структуры называются классами. Они определяются с помощью ключевого слова class.

Для переменных s1 и s2 возможно присваивание

s1=s2

так как эти переменные созданы на базе одного шаблона. После такого присваивания поля структуры s1 будут содержать ту же информацию, что и поля s2. Если мы опишем две структуры с одними и теми же полями, но первая структура будет иметь шаблон student1, а вторая student2, то присваивание s1=s2 недопустимо.

Битовые поля

Метод использования битовых полей для доступа к битам основан на структурах. Битовое поле, на самом деле, - это просто особый тип структуры, определяющей, какую длину имеет каждый член. Стандартный вид объявления битовых полей следующий:

struct имя структуры {
тип имя1: длина;
тип имя2: длина;
...
тип имяN: длина;
}

Битовые поля должны объявляться как int, unsigned или signed. Битовые поля длиной 1 должны объявляться как unsigned, поскольку 1 бит не может иметь знака. Битовые поля могут иметь длину от 1 до16 бит для 16-битных сред и от 1 до 32 бит для 32-битных сред. В Borland C++ самый левый бит является знаковым.

Объединение - это тоже пользовательский тип данных, который очень похож на структуру. Только тут все данные объединения занимают одну и туже область в памяти. Т.е. на каком-то этапе вам нужен один тип данных, на другом - другой. В общем, **объединение** экономит вашу память от ненужных на данном этапе переменных.

Так как объединение хранит и использует всегда одно поле их множества на выбор,

то возникает вопрос о выделении памяти под это поле. Тут принцип понятный - выбирается наибольший из типов данных.

объявление

Все то же самое, как и объявление структуры, только вместо специального слова struct используется **union**. Вот пример:

```
union chislo {
   int a;
   float b;
};
```

Разрешенные операции:

1. можно присваивать объединения друг другу
2. адрес брать так же ни кто не запрещал
3. к элементам можно получить доступ, так же как и в структурах, т.е. через (.) или (->)

Объединение (union) можно инициализировать только один значением, причем оно должно соответствовать первому элементу этого объединения. В нашем случае:

```
union chislo A = {34 }; // пойдет
union chislo B = {34.56 }; // нельзя
```

У нас первым элементом расположено поле int, поэтому при инициализации так же должно быть поле int.

Агрегат массива - это совокупность значений для каждого элемента массива. Использование агрегатов позволяет выполнять одновременное присваивание значений всем элементам массива в эффективной и элегантной форме.

Спецификаторы в объявлениях

Перед спецификаторами формата ввода ставится знак %, и они сообщают функции scanf() о типе далее читаемых данных. Эти спецификаторы перечислены в таблице. Спецификаторы формата рассматриваются слева направо, и в таком же порядке с ним сопоставляются аргументы из списка аргументов.

Таблица: Спецификаторы формата функции scanf()

Код	Значение
%c	Читает одиночные символы
%d	Читает десятичное число
%i	Читает десятичное число
%e	Читает число с плавающей запятой
%f	Читает число с плавающей запятой
%g	Читает число с плавающей запятой
%o	Читает восьмеричное число
%s	Читает строку
%x	Читает шестнадцатиричное число
%p	Читает указатель
%n	Получает целочисленное значение, равное числу прочитанных символов
%u	Читает беззнаковое целое
%[]	Сканирует множество символов

Модификаторы в объявлениях

За исключением типа void, основные типы данных могут иметь различные модификаторы. Модификаторы используются для более точного управления ситуацией. Ниже приведен список модификаторов:
signed
unsigned
short

Модификаторы signed, unsigned, long и short могут применяться к целочисленным типам. К символам можно применять signed и unsigned, long может применяться к типу double. Таблица показывает все допустимые комбинации стандарта ANSI C для 16-битных типов данных вместе с размером типа в битах и границами применения в Borland C++.

Typedef

Язык С позволяет определять имена новых типов данных с помощью ключевого слова typedef. На самом деле здесь не создается новый тип данных, а определяется новое имя существующему типу. Он позволяет облегчить создание машинно-независимых программ. Единственное, что потребуется при переходе на другую платформу, - это изменить оператор typedef. Он также может помочь документировать код, позволяя назначать содержательные имена стандартным типам данных. Стандартный вид оператора typedef следующий:
typedef тип имя;
где тип — это любой существующий тип данных, а имя - это новое имя для данного типа. Новое имя определяется в дополнение к существующему имени типа, а не замещает его. Например, можно создать новое имя для float, используя
typedef float balance;
Данный оператор сообщает компилятору о необходимости распознавания balance как другого имени для float. Далее можно создать вещественную переменную, используя balance:

balance past_due;
Здесь past_due - это вещественная переменная типа balance, другими словами - типа float. Можно использовать typedef для создания имен для более сложных типов. Например:
typedef struct {
float due;
int over_due;
char name[40];
} client; /* здесь client - это имя нового типа */
client clist[NUM_CLIENTS]; /* определение массива структур типа client */

Использование typedef может помочь при создании более легкого для чтения и более переносимого кода. Но надо помнить, что на самом деле не создаются никакие новые типы данных.