Лекции по C/C++: составные типы данных (перечисления, объединения, структуры структур)

Вторым стандартным составным типом данных является *перечисление*. Перечисление определяет список целочисленных значений, которые может принимать переменная и позволяет присвоить этим значениям "названия". Например, определим именованное перечисление `mode` ("режим работы" чего-то в программе) и укажем его возможные состояния:

```
enum mode { LAZY, WAIT, BUZY, ERROR };
```

Теперь в программе можно создать переменную, имеющую тип перечисления `mode` и использовать её:

```
mode mymode = LAZY;
//...
switch (mymode) {
 case LAZY: /* ... */ break;
 case WAIT: /* ... */ break;
 /* ... */
}
```

Здесь "читабельные" значения `WAIT` или `BUZY` напомнят о том, какую информацию сейчас хранит переменная, гораздо лучше, чем безликие числовые значения `1` или `2`.

Все константы перечисления должны быть допустимыми идентификаторами C++, обычно, как и для других констант, имена пишут БОЛЬШИМИ буквами (это соглашение, а не правило языка).

Тип `enum`, как правило, интерпретируется компилятором как `int` (или `unsigned`), первый элемент списка по умолчанию равен `0`, если не указано иного.

Основное применение перечислений – удобные мнемонические имена для переменных, имеющих фиксированный набор значений, таких, как цвета для рисования, дни недели, месяцы или сезоны года, состояния автомата, социальные статусы человека и т.д.

Основные недостатки перечислений - отсутствие встроенных средств для вывода строковых значений констант перечисления, а также не всегда очевидная итерация по членам перечисления.

Приведём пример:

```
enum day {
 SATURDAY, SUNDAY=0, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY
}; //SATURDAY также =0!
day today = SATURDAY;
cout << today << endl; //0
```

```
//today++; //ошибка компиляции – к типу неприменимо ++
today = FRIDAY;
cout << today << endl; //5
int workday = SATURDAY;
for (int i=0; i<7; i++) {
 cout << workday++ << " ";
 //теперь ++ работает, но напечатается 0 1 ... 6
}
```

Наконец, к составным типам данных можно отнести и *объединение* (`union`), позволяющее интерпретировать один и тот же объект (одну и ту же память) как данные различных типов:

```
union uchar {
 char c[2];
 unsigned short int u;
};
```

Здесь элемент типа `uchar`, в зависимости от того, для чего он нам нужен, может быть интерпретирован и как массив из двух символов типа `char`, и как беззнаковое целое 2-байтовое значение:

```
cout << sizeof(uchar) << endl; //2
uchar n;
n.u=0x6121;
cout << n.c[0] << n.c[1] << endl; //!a
cout << n.u << endl; //24865
```

Контроль за соответствием размерности полей в `union` возлагается на программиста, следует учитывать, что компилятор может выравнивать данные на границу слова или двойного слова.

Нечасто используемой в прикладном программировании, но полезной возможностью структур являются *битовые поля*. Они позволяют "упаковать" поле структуры в указанное количество битов вместо байтов:

```
тип идентификатор: константное_выражение;
```

Целочисленное *константное_выражение*, записанное после двоеточия, определяет число битов, выделяемых под переменную.

Указанный в шаблоне *идентификатор* необязателен. Неименованное битовое поле означает пропуск указанного числа битов перед размещением следующего элемента структуры. Неименованное битовое поле, для которого указан размер `0`, имеет специальное назначение: оно гарантирует, что память для следующей переменной в этой структуре будет начинаться на границе машинного слова (размер машинного слова зависит от архитектуры ЭВМ и обычно равен разрядности регистров процессора). Это относится и к следующему битовому полю.

Битовое поле не может выходить за границу ячейки памяти объявленного для него типа. Например, битовое поле `unsigned int` либо упаковывается в пространство, оставшееся в текущей ячейке размерностью `sizeof(unsigned int)*8` битов от размещения предыдущего битового поля, либо, если предыдущий элемент структуры

не был битовым полем или памяти в текущей ячейке недостаточно, в новую ячейку `unsigned int`.

На практике битовые поля используются обычно в двух целях:

- для экономии памяти, поскольку позволяют плотно упаковать значения не по границам байтов;
- в системном программировании, например, для организации удобного доступа к регистрам внешних устройств, в которых различные биты могут иметь самостоятельное функциональное назначение, при доступе к элементам файловых таблиц и т.п.

В качестве примера покажем структуру, описывающую отдельную позицию экрана текстовой консоли и дающую возможность доступа к отдельным свойствам экранной позиции через битовые поля.

```
struct texel {
 int background: 8; //фон
 int color: 4; //цвет
 int underline: 1; //подчёркивание
 int blink: 1; //мерцание
};
union bits {
 texel t;
 int d;
};
```

Объединение `bits` позволит манипулировать со структурой как с обычным целым числом, в частности, выводить его на экран консоли в виде цепочки бит. Для работы показанного ниже кода нужно подключить стандартное пространство имён и библиотеку `bitset`:

```
#include <bitset>
using namespace std;
```

Класс `bitset`, доступный в современных стандартах C++, удобен для решения задач, связанных с манипулированием отдельными битами или с булевой алгеброй.

```
texel t;
 cout << "Size of texel = "  << sizeof(texel) << " byte(s)" << endl;
 cout << "Size of bits = " << sizeof(bits) << " byte(s)" << endl;
 t.background = 0xFF;
 t.color = 0;
 t.underline = 1;
 t.blink = 0;
 bits b; b.t = t;
 cout << "T=" << hex << b.d << " (hex)" << endl;
 cout << "T=" << bitset<sizeof(b.d)*8>(b.d) << " (bin)" << endl;
```

Вывод этой программы на 32-разрядном компьютере получился таким:

```
Size of texel = 4 byte(s)
Size of bits = 4 byte(s)
```

```
T=ccccd0ff (hex)
T=11001100110011001101000011111111 (bin)
```

Обратите внимание, что неиспользуемые в структуре биты могут оказаться заполненными "мусором".

В качестве законченного примера и "ступенчатого" моделирования структур, включающих в себя другие структурные типы, можно разобрать вот эту задачу.

Оглавление серии

теги: c++ учебное

Здесь можно оставить комментарий, обязательны только выделенные цветом поля. Не пишите лишнего, и всё будет хорошо

Ваше имя:

Пароль (если желаете запомнить имя):

Любимый URL (если указываете, то вставьте полностью):

Текст сообщения (до 1024 символов):

Введите код сообщения: **3**3_{9_1}

Добавить Сброс

29.11.2015, 00:14; рейтинг: 4905