Перечисляемые типы

ENUM

- Иногда, удобной заменой константам является объявление специального ПЕРЕЧИСЛИМОГО ТИПА, **ENUM**, который содержит набор перечислимых констант.

ENUM

```
enum Months {
    JAN = 1, FEB, MAR, APR, MAY, YUN, YUL, AUG, SEP, OCT, NOV, DEC
};
short month;
Months yourMonth;
cin >> month;
yourMonth = (Months)month;
switch (yourMonth) {
    case JAN:
    cout << "Январь";
    break;
    case FEB:
    cout << "Февраль";
    break;
}
cout << endl;
```

ENUM

```
enum Months {
    JAN = 1, FEB, MAR, APR, MAY, YUN, YUL, AUG, SEP, OCT, NOV, DEC
};
short month;
cin >> month;
switch ((Months)month) {
    case JAN:
    cout << "Январь";
    break;
    case FEB:
    cout << "Февраль";
    break;
}
cout << endl;
```

ENUM

```
enum Months {
    JAN = 1, FEB, MAR, APR, MAY, YUN, YUL, AUG, SEP, OCT, NOV, DEC
};
Months yourMonth;
yourMonth  = MAR;
switch (yourMonth) {
    case JAN:
    cout << "Январь";
    break;
    case FEB:
    cout << "Февраль";
    break;
}
cout << endl;
```

ENUM

```
enum Months {
    JAN = 1, FEB, MAR, APR, MAY, YUN, YUL, AUG, SEP, OCT, NOV, DEC
};
Months yourMonth;
yourMonth  = YUN;
switch (yourMonth) {
    case JAN:
    cout << "Январь";
    break;
    case FEB:
    cout << "Февраль";
    break;
}
cout << endl;
```

ENUM

- Перечислимый тип вводится ключевым словом enum и задает набор значений, определяемый пользователем.
- Набор значений заключается в фигурные скобки и является набором целых именованных констант, представленных своими идентификаторами.
- Эти константы называются перечислимыми константами.

ENUM

- enum Months {JAN = 1, FEB, MAR, APR, MAY, YUN, YUL, AUG, SEP, OCT, NOV, DEC};

- Это объявление создает определенный пользователем тип Months с константами перечисления, представляющими месяцы года.

- Поскольку первое значение приведенного перечисления установлено равным 1, оставшиеся значения увеличиваются на 1 от 1 до 12.

- В объявлении перечислимого типа любой константе перечисления можно присвоить целое значение.

ENUM

Типичная ошибка:

- После того, как константа перечисления определена, попытка присвоить ей другое значение является синтаксической ошибкой.

ENUM

Правила хорошего тона:

- Наименование перечислимого типа следует писать прописными буквами, первая буква пишется заглавной (в отличии от переменных), слова пишутся слитно, разделение слов осуществляется верблюжьим стилем.
- Идентификаторы константных значений перечислений пишут ЗАГЛАВНЫМИ буквами, слова разделяются нижней чертой _ как имена констант.

Пример: TestEnum {TEST_NAME = 1, SUPER_TEST = 2}

Основные моменты перечислений

1. Использование перечислений вместо целых констант облегчает чтение программы.
2. Идентификаторы в enum должны быть уникальными, но отдельные константы перечисления могут иметь одинаковые **целые** значения.
3. Константы перечисления могут иметь **только целые значения**.
4. Набор идентификаторов перечислимого типа — собственный уникальный тип, отличающийся от других целочисленных типов.

Основные моменты перечислений

5. Перечислимые константы могут определяться и инициализироваться произвольными целочисленными константами, а также константными выражениями:

```
enum Ages {MILTON= 47, IRA, GOGA= 56, PHILIP = GOGA + 7};
```

6. Каждое перечисление является отдельным типом. Типом элемента перечисления является само перечисление. Например, в

```
enum Keyword {ASM, AUTO, BREAK};
```

Значение AUTO принадлежит типу **Keyword**

```
Keyword testWord;
```

Переменная **testWord** имеет тип **Keyword** и способна принимать только 3 значения: ASM, AUTO или BREAK; Например:

```
testWord = ASM;
```

Основные моменты перечислений

6. Перечислимая константа может быть объявлена анонимно, то есть без имени типа.

```
enum {FALSE, TRUE};
enum {LAZY, HAZY, CRAZY} why;
```

- Первое объявление — распространенный способ объявления мнемонических целочисленных констант.
- Второе объявление объявляет переменную перечислимого типа why, с допустимыми значениями этой переменной lazy, hazy и crazy

Основные моменты перечислений

7. Перечисления могут неявно преобразовываться в обычные целочисленные типы, но не наоборот.

```
enum Boolean {FALSE, TRUE} q;
enum Signal {OFF, ON} a = ON; // a инициализируется в on enum
int i, j = TRUE; // верно true преобразуется в 1
a = OFF; // верно
i = a; // верно i становится 1
q = a; // неверно два различных типа
q = (Boolean)a; // верно явное преобразование (приведение)
```

Спасибо за внимание.