

Ravesli [Ravesli](#)


- [Уроки по C++](#)
- [OpenGL](#)
- [SFML](#)
- [Qt5](#)
- [RegEx](#)
- [Ассемблер](#)
- [Купить .PDF](#)


## Урок №58. Перечисления

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 15 Окт 2020 |

 74567

[↑](#)  31

Язык C++ позволяет программистам создавать свои собственные (пользовательские) типы данных.

Оглавление:

1. [Перечисляемые типы](#)
2. [Имена перечислений](#)
3. [Значения перечислителей](#)
4. [Обработка перечислений](#)
5. [Вывод перечислителей](#)
6. [Выделение памяти для перечислений](#)
7. [Польза от перечислений](#)
8. [Тест](#)

## Перечисляемые типы

**Перечисление** (или «*перечисляемый тип*») — это тип данных, где любое значение (или «*перечислитель*») определяется как **символьная константа**. Объявить перечисление можно с помощью ключевого слова `enum`. Например:

```
1 // Объявляем новое перечисление Colors
2 enum Colors
3 {
4     // Ниже находятся перечислители - все возможные значения этого типа данных
5     // Каждый перечислитель отделяется запятой (НЕ точкой с запятой)
6 }
```

```
7   COLOR_RED,  
8   COLOR_BROWN,  
9   COLOR_GRAY,  
10  COLOR_WHITE,  
11  COLOR_PINK,  
12  COLOR_ORANGE,  
13  COLOR_BLUE,  
14  COLOR_PURPLE, // о конечной запятой читайте ниже  
15 }; // однако сам enum должен заканчиваться точкой с запятой  
16  
17 // Определяем несколько переменных перечисляемого типа Colors  
18 Colors paint = COLOR_RED;  
   Colors house(COLOR_GRAY);
```

Объявление перечислений не требует выделения памяти. Только когда переменная перечисляемого типа определена (например, как переменная `paint` в примере, приведенном выше), только тогда выделяется память для этой переменной.

Обратите внимание, каждый перечислитель отделяется запятой, а само перечисление заканчивается точкой с запятой.

До C++11, конечная запятая после последнего перечислителя (как после `COLOR_PURPLE` в примере, приведенном выше) не разрешается (хотя многие компиляторы её все равно принимают). Однако начиная с C++11 конечная запятая разрешена.

## Имена перечислений

Идентификаторы перечислений часто начинаются с заглавной буквы, а имена перечислителей вообще состоят только из заглавных букв. Поскольку перечислители вместе с перечислением находятся в едином пространстве имен, то имена перечислителей не могут повторяться в разных перечислениях:

```
1  enum Colors  
2  {  
3      YELLOW,  
4      BLACK, // BLACK находится в глобальном пространстве имен  
5      PINK  
6  };  
7  
8  enum Feelings  
9  {  
10     SAD,  
11     ANGRY,  
12     BLACK // получим ошибку, так как BLACK уже используется в enum Colors  
13 };
```

Распространено добавление названия перечисления в качестве префикса к перечислителям, например: `ANIMAL_` или `COLOR_`, как для предотвращения конфликтов имен, так и в целях комментирования кода.

## Значения перечислителей

Каждому перечислителю автоматически присваивается целочисленное значение в зависимости от его позиции в списке перечисления. По умолчанию, первому перечислителю присваивается целое число 0, а каждому следующему — на единицу больше, чем предыдущему:

```
1 #include <iostream>
2
3 enum Colors
4 {
5     COLOR_YELLOW, // присваивается 0
6     COLOR_WHITE, // присваивается 1
7     COLOR_ORANGE, // присваивается 2
8     COLOR_GREEN, // присваивается 3
9     COLOR_RED, // присваивается 4
10    COLOR_GRAY, // присваивается 5
11    COLOR_PURPLE, // присваивается 6
12    COLOR_BROWN // присваивается 7
13 };
14
15 int main()
16 {
17     Colors paint(COLOR_RED);
18     std::cout << paint;
19
20     return 0;
21 }
```

Результат выполнения программы:

4

Можно и самому определять значения перечислителей. Они могут быть как положительными, так и отрицательными, или вообще иметь аналогичные другим перечислителям значения. Любые, не определенные вами перечислители, будут иметь значения на единицу больше, чем значения предыдущих перечислителей. Например:

```
1 // Определяем новый перечисляемый тип Animals
2 enum Animals
3 {
4     ANIMAL_PIG = -4,
5     ANIMAL_LION, // присваивается -3
6     ANIMAL_CAT, // присваивается -2
7     ANIMAL_HORSE = 6,
8     ANIMAL_ZEBRA = 6, // имеет то же значение, что и ANIMAL_HORSE
9     ANIMAL_COW // присваивается 7
10 };
```

Обратите внимание, `ANIMAL_HORSE` и `ANIMAL_ZEBRA` имеют одинаковые значения. Хотя C++ это не запрещает, присваивать одно значение нескольким перечислителям в одном перечислении не рекомендуется.

**Совет:** Не присваивайте свои значения перечислителям.

**Правило:** Не присваивайте одинаковые значения двум перечислителям в одном перечислении, если на это нет веской причины.

## Обработка перечислений

Поскольку значениями перечислителей являются целые числа, то их можно присваивать целочисленным переменным, а также выводить в консоль (как переменные [типа int](#)):

```
1  #include <iostream>
2
3  // Определяем новый перечисляемый тип Animals
4  enum Animals
5  {
6      ANIMAL_PIG = -4,
7      ANIMAL_LION, // присваивается -3
8      ANIMAL_CAT, // присваивается -2
9      ANIMAL_HORSE = 6,
10     ANIMAL_ZEBRA = 6, // имеет то же значение, что и ANIMAL_HORSE
11     ANIMAL_COW // присваивается 7
12 };
13
14 int main()
15 {
16     int mypet = ANIMAL_PIG;
17     std::cout << ANIMAL_HORSE; // конвертируется в int, а затем выводится на экран
18
19     return 0;
20 }
```

Результат выполнения программы:

6

Компилятор не будет [неявно конвертировать](#) целочисленное значение в значение перечислителя. Следующее вызовет ошибку компиляции:

```
1 Animals animal = 7; // приведет к ошибке компиляции
```

Тем не менее, вы можете сделать подобное с помощью [оператора static\\_cast](#):

```
1 Colors color = static_cast<Colors>(5); // но делать так не рекомендуется
```

Компилятор также не позволит вам вводить перечислители через `std::cin`:

```
1 #include <iostream>
2
3 enum Colors
4 {
5     COLOR_PURPLE, // присваивается 0
6     COLOR_GRAY, // присваивается 1
7     COLOR_BLUE, // присваивается 2
8     COLOR_GREEN, // присваивается 3
9     COLOR_BROWN, // присваивается 4
10    COLOR_PINK, // присваивается 5
11    COLOR_YELLOW, // присваивается 6
12    COLOR_MAGENTA // присваивается 7
13 };
14
15 int main()
16 {
17     Colors color;
18     std::cin >> color; // приведет к ошибке компиляции
19
20     return 0;
21 }
```

Однако, вы можете ввести целое число, а затем использовать оператор `static_cast`, чтобы поместить целочисленное значение в перечисляемый тип:

```
1 int inputColor;
2 std::cin >> inputColor;
3
4 Colors color = static_cast<Colors>(inputColor);
```

Каждый перечисляемый тип считается отдельным типом. Следовательно, попытка присвоить перечислитель из одного перечисления перечислителю из другого — вызовет ошибку компиляции:

```
1 Animals animal = COLOR_BLUE; // приведет к ошибке компиляции
```

Как и в случае с константами, перечисления отображаются в [отладчике](#), что делает их еще более полезными.

## Вывод перечислителей

Попытка вывести перечисляемое значение с помощью `std::cout` приведет к выводу целочисленного значения самого перечислителя (т.е. его порядкового номера). Но как вывести значение перечислителя в виде текста? Один из способов — написать функцию с использованием стейтментов `if`:

```
1 enum Colors
2 {
3     COLOR_PURPLE, // присваивается 0
```

```
4    COLOR_GRAY, // присваивается 1
5    COLOR_BLUE, // присваивается 2
6    COLOR_GREEN, // присваивается 3
7    COLOR_BROWN, // присваивается 4
8    COLOR_PINK, // присваивается 5
9    COLOR_YELLOW, // присваивается 6
10   COLOR_MAGENTA // присваивается 7
11 };
12
13 void printColor(Colors color)
14 {
15     if (color == COLOR_PURPLE)
16         std::cout << "Purple";
17     else if (color == COLOR_GRAY)
18         std::cout << "Gray";
19     else if (color == COLOR_BLUE)
20         std::cout << "Blue";
21     else if (color == COLOR_GREEN)
22         std::cout << "Green";
23     else if (color == COLOR_BROWN)
24         std::cout << "Brown";
25     else if (color == COLOR_PINK)
26         std::cout << "Pink";
27     else if (color == COLOR_YELLOW)
28         std::cout << "Yellow";
29     else if (color == COLOR_MAGENTA)
30         std::cout << "Magenta";
31     else
32         std::cout << "Who knows!";
33 }
```

## Выделение памяти для перечислений

Перечисляемые типы считаются частью семейства целочисленных типов, и компилятор сам определяет, сколько памяти выделять для переменных типа `enum`. По стандарту C++ размер перечисления должен быть достаточно большим, чтобы иметь возможность вместить все перечислители. Но чаще всего размеры переменных `enum` будут такими же, как и размеры обычных переменных типа `int`.

Поскольку компилятору нужно знать, сколько памяти выделять для перечисления, то использовать [предварительное объявление](#) с ним вы не сможете. Однако существует простой обходной путь. Поскольку определение перечисления само по себе не требует выделения памяти и, если перечисление необходимо использовать в нескольких файлах, его можно определить в [заголовочном файле](#) и подключать этот файл везде, где необходимо использовать перечисление.

## Польза от перечислений

Перечисляемые типы невероятно полезны для документации кода и улучшения читабельности.

Например, функции часто возвращают целые числа обратно в caller в качестве кодов ошибок, если что-то пошло не так. Как правило, небольшие отрицательные числа используются для представления возможных кодов ошибок. Например:

```
1 int readFileContents()
2 {
3     if (!openFile())
4         return -1;
5     if (!parseFile())
6         return -2;
7     if (!readFile())
8         return -3;
9
10    return 0; // если всё прошло успешно
11 }
```

Однако **магические числа**, как в вышеприведенном примере, не очень эффективное решение. Альтернатива — использовать перечисления:

```
1 enum ParseResult
2 {
3     SUCCESS = 0,
4     ERROR_OPENING_FILE = -1,
5     ERROR_PARSING_FILE = -2,
6     ERROR_READING_FILE = -3
7 };
8
9 ParseResult readFileContents()
10 {
11     if (!openFile())
12         return ERROR_OPENING_FILE;
13     if (!parseFile())
14         return ERROR_PARSING_FILE;
15     if (!readfile())
16         return ERROR_READING_FILE;
17
18     return SUCCESS; // если всё прошло успешно
19 }
```

Это и читать легче, и понять проще. Кроме того, функция, которая вызывает другую функцию, может проверить возвращаемое значение на соответствующий перечислитель. Это лучше, нежели самому сравнивать возвращаемый результат с конкретными целочисленными значениями, чтобы понять какая именно ошибка произошла, не так ли? Например:

```
1 if (readFileContents() == SUCCESS)
2 {
3     // Делаем что-нибудь
```

```
4     }
5     else
6     {
7         // Выводим сообщение об ошибке
8     }
```

Перечисляемые типы лучше всего использовать при определении набора связанных идентификаторов. Например, предположим, что вы пишете игру, в которой игрок может иметь один предмет, но этот предмет может быть нескольких разных типов:

```
1  #include <iostream>
2  #include <string>
3
4  enum ItemType
5  {
6      ITEMTYPE_GUN,
7      ITEMTYPE_ARBALET,
8      ITEMTYPE_SWORD
9  };
10
11 std::string getItemName(ItemType itemType)
12 {
13     if (itemType == ITEMTYPE_GUN)
14         return std::string("Gun");
15     if (itemType == ITEMTYPE_ARBALET)
16         return std::string("Arbalet");
17     if (itemType == ITEMTYPE_SWORD)
18         return std::string("Sword");
19 }
20
21 int main()
22 {
23     // ItemType - это перечисляемый тип, который мы объявили выше.
24     // itemType (с маленькой i) - это имя переменной, которую мы определяем ниже (типа
25     // ITEMTYPE_GUN - это значение перечислителя, которое мы присваиваем переменной itemType
26     ItemType itemType(ITEMTYPE_GUN);
27
28     std::cout << "You are carrying a " << getItemName(itemType) << "\n";
29
30     return 0;
31 }
```

Или, если вы пишете функцию для сортировки группы значений:

```
1  enum SortType
2  {
3      SORTTYPE_FORWARD,
4      SORTTYPE_BACKWARDS
5  };
```



```
6 |
7 | void sortData(SortType type)
8 | {
9 |     if (type == SORTTYPE_FORWARD)
10 |         // Сортировка данных в одном порядке
11 |     else if (type == SORTTYPE_BACKWARDS)
12 |         // Сортировка данных в обратном порядке
13 | }
```

Многие языки программирования используют перечисления для определения логических значений. По сути, **логический тип данных** — это простое перечисление всего лишь с двумя перечислителями: true и false! Однако в языке C++ значения true и false определены как ключевые слова вместо перечислителей.

## Тест

### Задание №1

Напишите перечисление со следующими перечислителями: ogre, goblin, skeleton, orc и troll.

#### Ответ №1

```
1 | enum MonsterType
2 | {
3 |     MONSTER_OGRE,
4 |     MONSTER_GOBLIN,
5 |     MONSTER_SKELETON,
6 |     MONSTER_ORC,
7 |     MONSTER_TROLL
8 | };
```

### Задание №2

Объявите переменную перечисляемого типа, который вы определили в задании №1, и присвойте ей значение ogre.

#### Ответ №2

```
1 | MonsterType eMonsterType = MONSTER_OGRE;
```

### Задание №3

**Правда или ложь:**

Перечислителям можно:

➔ присваивать целочисленные значения;

- не присваивать значения;
- явно присваивать значения типа с плавающей точкой;
- присваивать значения предыдущих перечислителей (например, `COLOR_BLUE = COLOR_GRAY`).

Перечислители могут быть:

- отрицательными;
- не уникальными.

### Ответ №3

Перечислителям можно:

- Правда.
- Правда. Перечислителю без значения будет неявно присвоено целочисленное значение предыдущего перечислителя +1. Если предыдущего перечислителя нет, то тогда присвоится значение 0.
- Ложь.
- Правда. Поскольку значениями перечислителей являются целые числа, а целые числа можно присвоить перечислителям, то одни перечислители могут быть присвоены другим перечислителям (хотя этого лучше избегать).

Перечислители могут быть:

- Правда.
- Правда.

Оценить статью:

★★★★★ (288 оценок, среднее: 4,86 из 5)



← [Урок №57. Введение в std::string](#)

[Урок №59. Классы enum](#)



## Комментариев: 31



1. *Aleksandr P:*

[14 июня 2020 в 15:24](#)

Доброго времени суток, в чем смысл указывать в return std::string, все прекрасно работает и без него, либо я чего-то не понимаю. Сама функция же имеет тип возвращаемого значения string.

```
1 std::string getItemName(ItemType itemType)
2 {
3     if (itemType == ITEMTYPE_GUN)
4         return std::string("Gun");    // здесь работает и return "Gun";
5     if (itemType == ITEMTYPE_ARBALET)
6         return std::string("Arbalet");
7     if (itemType == ITEMTYPE_SWORD)
8         return std::string("Sword");
9 }
```

[Ответить](#)

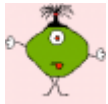


1. *Артурка:*

[28 октября 2020 в 21:08](#)

Да, ты прав, функция имеет возвращаемый тип std::string, по этому если не использовать конструкторы типа std::string в возвращаемом типе, то как const char\* в std::string будет произведен неявно. Видимо это сделано для наглядности происходящего.

[Ответить](#)



2. *Максим:*

[31 декабря 2019 в 06:57](#)

Возможно ли вывести через оператор std::cout на экран не значение одного из перечислителей а сам перечислитель? ( не значение цвета RED = 1, в виде единицы а само RED )

[Ответить](#)



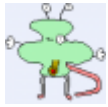
1. *Aleksandr P:*

[14 июня 2020 в 14:57](#)

С помощью Switch или If else;

```
1 switch (color) //переменная color
2 {
3     case COLOR_GRAY:
4         std::cout << "Gray";
5         break;
6     case COLOR_PINK:
7         std::cout << "Pink";
8         break;
9     case COLOR_BLUE:
```

```
10     std::cout << "Blue";
11     break;
12     case COLOR_PURPLE:
13         std::cout << "Purple";
14         break;
15     case COLOR_RED:
16         std::cout << "Red";
17         break;
18     default:
19         std::cout << "Unknown";
20         break;
21 }
```

[Ответить](#)

3. *Юлия:*  
[7 декабря 2019 в 21:50](#)

Очень мотивируют живые примеры с оружием и монстрами, а не с поднадоевшими студентами

[Ответить](#)

4. *Владимир:*  
[2 декабря 2019 в 18:32](#)

Прочитав урок 3 раза и туго понимая что к чему, я таки нашел ключевую фразу, которая сразу прояснила все в голове. Все стало понятно. Думаю не я один такой, поэтому рассказываю: для меня этой ключевой фразой послужил 3-ий сверху комментарий в 1-ом коде урока. // Это все возможные значения этого типа данных

...Шит!... это же ТИП данных, такой же как `int` или `char`! А перечисление — это буквально ПЕРЕЧИСЛЕНИЕ всех символьных имен этого( создаваемого нами самими типа), с присвоением им порядковых, или иных, на выбор, номеров! Т.е. просто перечисляем по очереди имена всех возможных значений этого типа, и присваиваем им ( или это делает автоматически компилятор) целочисленные идентификационные значения (номера). И конечно, они могут быть только константными(постоянными). Это же логично. К примеру, в типе `CHAR`, мы же заранее знаем все символьные значения и их числовые эквиваленты, и все это неизменно. А здесь у нас свобода назвать свой тип данных, выбрать символы и их числовые эквиваленты. Кстати, получается таблица ASCII с ее символами и нумерацией, это тоже своего рода перечисление, только по умолчанию. Вот и все!

[Ответить](#)

5. *koresh:*  
[5 августа 2019 в 17:38](#)

Перечисления перечислениями, а мне непонятно одно — для чего нужны переменные типа перечисления если и без них всё работает. Ответа нигде не нашёл.

[Ответить](#)1. *Алекс:*[6 июля 2020 в 19:35](#)

Перечисления нужны:

- 1) для лучшего комментирования кода;
- 2) для упрощения разработки.

Как минимум.

[Ответить](#)6. *Алексей:*[15 июля 2019 в 14:34](#)

Тест №3 — подстава, с плавающей ничего не было) Хотя я внимательно читал — целочисловые.

Вот присвоение значений других определителей — тоже самое, нету об этом, хотя что стоит дать число от другого, когда можно дать любое число перечислителю.

[Ответить](#)7. *Алексей:*[15 июля 2019 в 14:30](#)

В задании №2 сделал так:

```
1  #include <iostream>
2
3  enum Monsters
4  {
5  MONSTER_OGRE,
6  MONSTER_GOBLIN,
7  MONSTER_SKELETON,
8  MONSTER_ORC,
9  MONSTER_TROLL
10 };
11
12 int main()
13 {
14 Monsters mon(MONSTER_OGRE);
15 std::cout << mon << "\n";
16 }
```

Правда, тут отличается? Это же инициализация, не просто объявление.

[Ответить](#)



8. Денис:

[6 марта 2019 в 04:09](#)

Юрий, добрый день)

Я извиняюсь, но я вообще не могу понять зачем это нужно и где его применить... Как enum может помочь в сортировке и в прочем. Я учу каждый урок по очереди. На данном этапе вообще не понятно. Это очень не удобная конструкция (как по мне), скорее всего просто я её не понял. Если можно было-бы присвоить int, string и т.д. значения этим перечислениям, было-бы здорово, а так это просто растянутый текст типа (COLOR\_Space\_Blue), который хранит в себе (0,1,2 и в том духе далее), который, как написано, кроме как вывести это самое 0,1,2... , больше не на что не способно.

Может вы не раскрыли потенциал enum в статье, либо я тормоз и не могу понять что, как, а главное — зачем... Я уже 3-й раз возвращаюсь к этому уроку, но в пустую. Написал программу по выбору автомобиля, в зависимости от суммы, в надежде применить enum car, model, color... Но enum как не лепил, везде обхожусь самым if, cout, cin. Перечисления просто в шапке программы весит никому не нужное.

Ответьте пожалуйста, можно не выкладывать на сайт мой отзыв, почта у вас есть 😊

[Ответить](#)

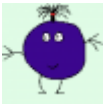


1. Константин:

[12 марта 2019 в 11:15](#)

Денис, "Приключения Электроника" смотрел? Помнишь Гусев в сердцах крикнул в сторону ЭВМ: "А-а-ах! Чурбан железный — чтоб ты заржавел!!!"? Вот и я о том же, что эта скотина никаких букв не понимает. Только цифры и только две. А эти enum выдуманы для напоминалок кодеру что какая цифра обозначает. Ну это как хозяйка семена весной замачивает-садит и пишет себе записочки где и какие семена сидят. А т.к. слова ПК не понимает, то приходится двойную бухгалтерию вести — сперва расписывать имена перечислений, а чтобы в консоли прочитав что это было пристёгивать к ним (с помощью cout) ещё одну записочку "САБЖ такой-то". А Юре не досуг на тупые вопросы отвечать — по себе знаю!

[Ответить](#)



2. Давид:

[21 марта 2020 в 22:33](#)

Преимущество исключительно в удобочитаемости кода. Я обычно перечисления использую как уникальные идентификаторы чего-либо, допустим горячей клавиши, когда читаешь код обработки горячих клавиш где вместо цифр 1, 2, 3... стоят имена из перечисления, то сразу становится понятно. Плюс, так относительно легко добавить новую горячую клавишу, не нужно следить за уникальностью идентификаторов.

[Ответить](#)



9. Alexey:

[4 февраля 2019 в 19:32](#)

Не знаю как в других компиляторах, но в VS очень удобно использовать Switch вместе с enum, он автоматом подставляет значения перечислений.

[Ответить](#)



10. Константин:

[4 января 2019 в 14:54](#)

Юра, эти уроки — настоящее интеллектуальное развлечение! Но вот я, таки, решил поиграть в игрушку-стрелялку. Запустил программу, т. е. режим run-time. Бегу, стреляю из ружья — разработчик дал мне его:

```
1 int main()
2 {
3     // ItemType - это перечисляемый тип, который мы объявили выше
4     // itemType (с маленькой i) - это имя переменной, которую мы определили (типа
5     // ITEMTYPE_GUN - это значение перечислителя, которое мы присвоили переменной
6     ItemType itemType(ITEMTYPE_GUN);
7
8     std::cout << "You are carrying a " << getItemName(itemType) << "\n";
9
10    return 0;
11 }
```

...и вот я подбегаю к такому участку, где нельзя шуметь, иначе на шум сбегутся столько монстров, что разорвут меня вместе с ружьём. А вот из арбалета можно тихонько завалить одного монстра-охранника и пройти это непроходимое место дальше. Только КАК можно на ходу переключиться на арбалет? Он же заявлен в типах вооружений:

```
1 enum ItemType
2 {
3     ITEMTYPE_GUN,
4     ITEMTYPE_ARBALET,
5     ITEMTYPE_SWORD
6 };
```

[Ответить](#)



1. Илья:

[26 февраля 2019 в 13:07](#)

поменять на арбалет:

```
1 itemType = ItemType (1);
```

или

```
1 itemType = ItemType(ITEMTYPE_ARBALET);
```

Ответить1. *Илья:*
[26 февраля 2019 в 13:31](#)

```
1 itemType = ITEMTYPE_SWORD;
```

Ответить2. *Константин:*
[1 марта 2019 в 22:14](#)

Ээ-э-эй, Илья! Когда я уже бегу внутри игры — это RUNTIME, т.е. режим выполнения проги процем. А ты предлагаешь вариант COMPILETIME, т.е. задать условия ДО начала игры...

Ответить1. *Денис:*
[26 ноября 2019 в 17:30](#)

можно сделать событием прокручивания колёсиком/клик на любую клавишу или комбинацию клавиш (так называемые hot keys), но это всё прелести Qt. Писал проэкты и много раз делал события на нажатия на кнопки клави, клацанье на кнопки мыши и т.д. и т.п.

Qt очень даже офигенная вещь в этом плане!

2. *Константин:*
[27 мая 2020 в 21:11](#)

А-а! ЛДогнал! Надо( используя статик\_кэст через промежуточную целочисленную переменную ) всунуть нужное значение в переменную, придуманного мною типа!

2. *Константин:*
[27 мая 2020 в 21:24](#)

```
1 cout << "Enter num of Weapon: 0 - GUN, 1 - ARBALET, 2 - SWORD ";
2 int inputWeapon{ enNum() };
3 ItemType itemType = static_cast<ItemType>(inputWeapon);
```



[Ответить](#)11. *Dimoss76:*[29 мая 2018 в 20:29](#)

Извините, не могу сообразить. Возникли вопросы:

1. можно ли имена перечислителей перебрать и вывести на печать с помощью цикла?
2. Можно ли переменной задать следующее значение из списка перечислителей?

[Ответить](#)1. *Юрий:*[8 июня 2018 в 15:32](#)

1. Можно, цикл `for` в помощь.
2. Тоже можно, инкремент в помощь.

[Ответить](#)12. *Liam:*[13 февраля 2018 в 10:10](#)

Ничего не понятно, в самом начале объявляем `enum`, но в "Выделение памяти для перечислений. Предварительное объявление" пишут, что объявлять предварительно нельзя.

Потом пишут, что объявления не требуют памяти, затем в абзаце написано, что и определение не требует памяти, так когда же память выделяется?

[Ответить](#)1. *Юрий:*[13 февраля 2018 в 18:01](#)

Объявление самого перечисления без объявления переменных не приводит к выделению памяти. Память выделяется при определении переменных перечисления, НЕ при объявлении самого перечисления — что тут непонятного и как объяснить еще проще — я уже и не знаю.

Насчет предварительного объявления — вы урок об этом читали? Вот [урок о предварительном объявлении](#). Объявление и предварительное объявление — это не одно и то же. Объявление следует вместе с определением (телом перечисления), предварительное объявление — это одна строчка кода.

Объявлять перечисление вы можете сколько угодно раз, использовать предварительное объявление (одну строчку кода, без определения самого перечисления — тела перечисления) — нет.

[Ответить](#)



1. *Liam:*

[14 февраля 2018 в 20:50](#)

Гхм, кто-то из нас явно что-то не так понимает, наверное я, раз я только начал изучать C++ . Посмотрел вашу статью, вы в примерах о предварительном объявлении и просто об объявлении даете абсолютно один и тот же пример —

`int add(int x, int y)` . И там и там одна строчка кода.

И вот как раз в определении вы дали правильный пример `int add(int x, int y) {return x+y}`

Во-вторых я только что проверил, мой компилятор не ругается на конструкцию `enum Colors{};` . Те я могу объявлять таким образом `enum`

[Ответить](#)



1. *Юрий:*

[15 февраля 2018 в 23:52](#)

Запустите у себя код:

```

1  #include "stdafx.h"
2  #include <iostream>
3  #include <string>
4
5  enum Animals{}; // прототип функции - предварительное объявление
6
7
8  int main()
9  {
10     std::cout << "You are carrying a " << Animals(ANIMAL_PIG) << "
11     return 0;
12 }
13
14 enum Animals // определение
15 {
16     ANIMAL_PIG = -4,
17     ANIMAL_LION, // присвоено -3
18     ANIMAL_CAT, // присвоено -2
19     ANIMAL_HORSE = 6,
20     ANIMAL_ZEBRA = 6, // имеет то же значение, что и ANIMAL_HORSE
21     ANIMAL_COW // присвоено 7
22 };

```

Не скомпилируется. Почему? Потому что нельзя использовать предварительное объявление для перечислений.

Определение функции включает в себя объявление:

```
1 enum Animals
```

и определение:

```
1 {
2     ANIMAL_PIG = -4,
3     ANIMAL_LION, // присвоено -3
4     ANIMAL_CAT, // присвоено -2
5     ANIMAL_HORSE = 6,
6     ANIMAL_ZEBRA = 6, // имеет то же значение, что и ANIMAL_HORSE
7     ANIMAL_COW // присвоено 7
8 };
```

Прототип функции (то же предварительное объявление) включает в себя только строку объявления объекта:

```
1 enum Animals{};
```

Прототип функции указывается выше `main()`, а само определение функции ниже `main()`. Здесь уже на примерах показал вам.



13. **Илья:**

[14 ноября 2017 в 07:26](#)

Здравствуйте.

Есть вопрос:

В примере с задачей про GUN, ARBALET и SWORD программа не работает как надо, ведь функция `getItemName` всегда будет возвращать ARBALET, так как он равен 1 (`GUN == 0`, `SWORD == 2`). Если в типе данных `enum ItemType` всем присвоить 1, то выдает первое значение — GUN.

Как это исправить? Спасибо.

[Ответить](#)



1. **Юрий:**

[17 ноября 2017 в 14:44](#)

В примере функция `getItemName` возвращает Gun, так как мы передаем аргумент `itemType`, которому, предварительно в строке 26 (`ItemType itemType(ITEMTYPE_GUN);`) установили значение `ITEMTYPE_GUN`. Функция `getItemName` не будет работать без передаваемого аргумента, т.е. мы сами определяем, что будет выводиться на экран, указывая один из элементов структуры `ItemType` (`ITEMTYPE_GUN`, `ITEMTYPE_ARBALET` или `ITEMTYPE_SWORD`). Члены `ItemType` имеют значения от 0 до 2, но определение того, что будет выводить `getItemName` осуществляется в `main`, когда мы указываем аргумент. Как вы уже изменили код, что у вас программа выводит что-либо другое, кроме того, что вы указываете в строке 26 (`ItemType itemType(ITEMTYPE_GUN);`) — это уже под вопросом.

[Ответить](#)



14. **Old G.B.:**

[15 октября 2017 в 23:16](#)

вот здесь я впервые понял что ничего не понял

[Ответить](#)

1.  *Юрий:*  
[15 октября 2017 в 23:37](#)

Не без этого. Попробуйте внимательнее перечитать этот урок, но уже не следующий день. Поищите дополнительную информацию в Интернете на эту тему, если уж никак не идёт.

[Ответить](#)

## Добавить комментарий

Ваш E-mail не будет опубликован. Обязательные поля помечены \*

Имя \*

Email \*

Комментарий





☐ Сохранить моё Имя и E-mail. Видеть комментарии, отправленные на модерацию

☐ Получать уведомления о новых комментариях по электронной почте. Вы можете [подписаться](#) без комментирования.

[TELEGRAM](#)  [КАНАЛ](#)

[ПАБЛИК](#) 

## ТОП СТАТЬИ

-  [Словарь программиста. Сленг, который должен знать каждый кодер](#)
-  [Урок №1. Введение в программирование](#)
-  [70+ бесплатных ресурсов для изучения программирования](#)
-  [Урок №1: Введение в создание игры «SameGame» на C++/MFC](#)

-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

- [Ravesli](#)
- - [О проекте/Контакты](#) -
- - [Пользовательское Соглашение](#) -
- - [Все статьи](#) -
- Copyright © 2015 - 2020