Область видимости переменных и типы памяти. Пространства имён

Владислав Хорев
Ведущий программист, Luxoft

Владислав Хорев

О спикере:

- Ведущий программист в компании Luxoft
- Работает в IT с 2011 года
- Опыт разработки на C++ более 10 лет

Вспоминаем прошлое занятие

Вопрос: как хранятся переменные в памяти?

Вспоминаем прошлое занятие

Вопрос: как хранятся переменные в памяти?

Ответ: память состоит из ячеек фиксированного размера, для хранения переменной выделяется одна или несколько ячеек, где и хранится значение, к которому мы получаем доступ через переменную. Каждая ячейка имеет свой адрес — шестнадцатеричное число

Вспоминаем прошлое занятие

Вопрос: как узнать адрес переменной в памяти?

Вспоминаем прошлое занятие

Вопрос: как узнать адрес переменной в памяти?

Ответ: использовать оператор & перед названием переменной

Вспоминаем прошлое занятие

Вопрос: что происходит при простой передаче аргументов в функцию?

Вспоминаем прошлое занятие

Вопрос: что происходит при простой передаче аргументов в функцию?

Ответ: для каждого аргумента в функции создаётся новая переменная, и в эти переменные копируются значения, которые были переданы в функцию в качестве аргументов

Вспоминаем прошлое занятие

Вопрос: что такое глобальная переменная?

Вспоминаем прошлое занятие

Вопрос: что такое глобальная переменная?

Ответ: переменная, объявленная вне функций и по этой причине доступная этим функциям

Вспоминаем прошлое занятие

Вопрос: как сделать, чтобы переменную можно было менять в нескольких функциях, которые вызывают друг друга?

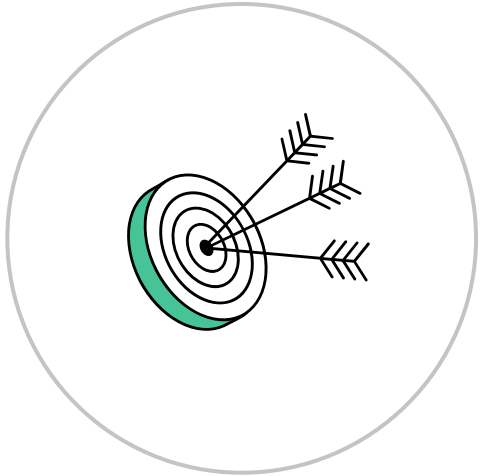
Вспоминаем прошлое занятие

Вопрос: как сделать, чтобы переменную можно было менять в нескольких функциях, которые вызывают друг друга?

Ответ: использовать глобальную переменную (плохой вариант) или использовать передачу переменных по ссылке (хороший вариант)

Цели занятия

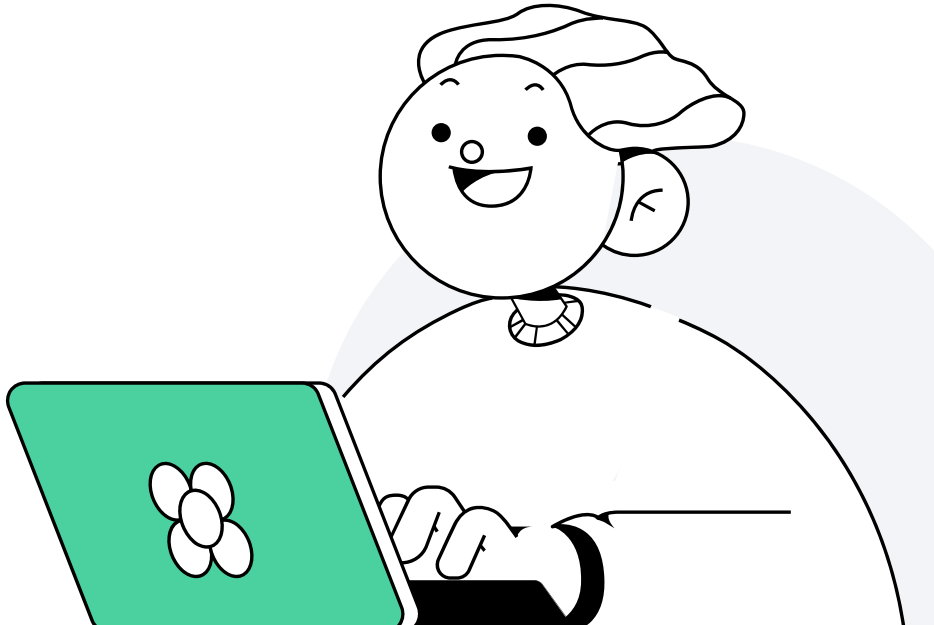
- Разберёмся с типами памяти в C++
- Посмотрим, что такое ключевое слово static
- Узнаем, что такое пространства имён
- Выясним, как пользоваться ключевым словом namespace



План занятия

1. Типы памяти
2. Статическая память
3. Автоматическая память
4. Динамическая память
5. Пространства имён
6. Итоги
7. Домашнее задание

*Нажми на нужный раздел для перехода



Типы памяти

1

Типы памяти

На прошлой лекции мы с вами разобрались, как хранятся переменные в памяти. Оказывается, память в программах тоже бывает разная.

В программах существуют 3 вида памяти:

- Статическая
- Автоматическая
- Динамическая

Типы памяти

От вида памяти зависит, где она размещается, сколько её доступно, кто управляет этой памятью, когда она инициализируется.

Сегодня мы познакомимся со **статической** и **автоматической** памятью,
а динамическую память обрисуем вскользь, подробно же разберём
в следующих лекциях

Статическая память

1.1

Статическая память

Статическая память выделяется ещё до начала работы программы, на стадии компиляции и сборки.

Статические переменные имеют фиксированный адрес, известный до запуска программы и не изменяющийся в процессе её работы. Статические переменные создаются и инициализируются до входа в функцию main, с которой начинается выполнение программы

Статические переменные

На прошлом слайде мы упомянули статические переменные

Глобальные переменные

Ещё раз вспомним, что такое глобальные переменные. Глобальные переменные — переменные, определённые **вне функций,** в описании которых отсутствует слово static. Это будет важно далее

```
int glob_var = 100; // глобальная переменная, при создании будет иметь значение 100

void func()
{
    glob_var = 10; // присваиваем глобальной переменной значение 10
}

int main(int argc, char** argv)
{
    glob_var = 5; // присваиваем глобальной переменной значение 5
    func();
    return 0;
}
```

Статические переменные

Статические переменные — это переменные, в описании которых присутствует слово static.

Статические переменные подобны глобальным за одним исключением: область видимости статической переменной ограничена одним файлом, внутри которого она определена.

Статические переменные

```
void func()
{
    stat_var = 10;              // ???
}

static int stat_var = 100;      // статическая переменная, при создании будет иметь значение 100

int main(int argc, char** argv)
{
    stat_var = 5;               // присваиваем статической переменной значение 5
    func();
    std::cout << stat_var;      // 5
    return 0;
}
```

Что скажет компилятор?

Статические переменные

```
void func()
{
    stat_var = 10; // Ошибка!! Здесь переменная stat_var недоступна, так как объявлена ниже
по тексту
}

static int stat_var = 100; // статическая переменная, при создании будет иметь значение 100

int main(int argc, char** argv)
{
    stat_var = 5; // присваиваем статической переменной значение 5
    func();
    std::cout << stat_var; // 5
    return 0;
}
```

Статические переменные

Статическая переменная может быть описана и **внутри функции.** Такая переменная будет доступна только внутри этой функции, но будет создана всего один раз. Более того, такая переменная будет сохранять своё значение между различными вызовами функции

Напишем программу для демонстрации особенностей статической переменной внутри функции

Готовый пример кода

Статические функции

Кроме переменных, ключевое слово static можно использовать при объявлении функций. Область видимости такой функции будет ограничена файлом, в котором она была объявлена. То есть из других файлов ей воспользоваться будет нельзя

```
static void func()
{
    …
}

int main(int argc, char** argv)
{
    func();
    return 0;
}
```

Ключевое слово static

Мы выяснили, что ключевое слово static выполняет две различные функции, которые друг друга не исключают:

- Указание типа памяти, в которой размещается помеченный этим ключевым словом объект
- Ограничение области видимости

Со статической памятью мы разобрались, давайте перейдём к автоматической

Автоматическая память

1.2

Автоматическая память

В автоматической памяти размещаются последовательность вызванных функций и все те обычные переменные, которыми мы привыкли пользоваться.
Поэтому такие переменные называются **автоматическими**

Автоматическая память

Автоматическая память, как и статическая, тоже выделяется операционной системой при старте программы и ограничена по размеру. Однако автоматические переменные отличаются от статических — прежде всего временем жизни.

Статические переменные создаются **один раз** при старте программы и уничтожаются при завершении программы. Существование автоматических переменных ограничено их **областью видимости**

Напишем программу для демонстрации того, как создаются и уничтожаются локальные переменные

Готовый пример кода

Автоматическая память

Поведение написанной нам программы вызывает вопросы: почему,
если переменная i создаётся во втором цикле заново, у неё тот же адрес,
что и раньше?

Это зависит от компилятора программы. Например, если вы скомпилируете
исходный код этой программы в Visual Studio на машине с ОС Windows,
то вы получите другой результат.

Компилятор решает, как размещать переменные в памяти, и здесь он был
достаточно умён, чтобы переиспользовать память, выделенную под первую
переменную i, для второй переменной i. Но так может быть не всегда и не везде.
Подобное справедливо и для переменной x.

Динамическая память

1.3

Динамическая память

Динамическая память — это память, которой в рамках программы управляет сам программист.

Программист запрашивает у операционной системы столько памяти, сколько ему надо. Теоретически он ограничен размером оперативной памяти компьютера

Динамическая память

После использования программист должен **очистить** эту память, то есть сообщить операционной системе, что эта память ему больше не нужна.

Если программист этого не сделает, то произойдёт так называемая **утечка памяти** — ситуация, когда программист забыл освободить память, но больше не может ей воспользоваться, потому что перестал следить за ней. А ОС думает, что эта память ещё нужна программе, поэтому больше никто не может ей пользоваться

Пространства имён

2

Пространства имён

Пространства имён возникли как решение проблемы **конфликта имён.**
Конфликт имён — ситуация, когда вы хотите назвать переменную или функцию каким-то именем, но в процессе компиляции программы выясняется, что это имя уже используется.

Такое часто возникает в программах, состоящих из множества файлов.
Такие очевидные названия как sum или abs могут использоваться
в разных контекстах

Пространства имён

Пространство имён определяет область кода, в которой гарантируется уникальность всех идентификаторов. По умолчанию глобальные переменные и обычные функции определены в **глобальном пространстве имён.**

При этом язык C++ позволяет объявлять собственные пространства имён с помощью ключевого слова namespace

Пространства имён

Всё, что объявлено внутри пользовательского пространства имён, принадлежит только этому пространству имён, а не глобальному. При этом вы можете объявлять переменные и функции в одном пространстве имён, но в разных файлах: объявление пространства имён может существовать во многих экземплярах.

Чтобы использовать функцию или переменную, объявленную в другом пространстве имён, нужно либо указать это пространство имён с двумя двоеточиями после него (::), либо **подключить** это пространство имён

Пространства имён. Пример

```
namespace my_namespace // объявляем пространство имён my_namespace
{
    int value = 10; // переменная value будет доступна в пространстве имён my_namespace
}
namespace my_namespace // ещё раз объявляем пространство имён my_namespace
{
    int number = 4; // переменная number тоже будет доступна в пространстве имён my_namespace
}
int main(int argc, char** argv)
{
    my_namespace::value = 5; // используем переменную value из пространства имён my_namespace
    return 0;
}
```

Напишем программу со своими пространствами имён

Готовый пример кода

Пространства имён. Подключение

Чтобы подключить пространство имён, нужно использовать конструкцию **using namespace <пространство имён>.** Весь код, написанный после этой команды, может использовать переменные и функции, объявленные в этом пространстве имён, без указания самого пространства.

Этим механизмом нужно пользоваться с осторожностью. Его использование может привести к конфликту имён. Гораздо безопаснее всегда полностью указывать пространство имён переменной или функции, которой вы хотите воспользоваться

Пространства имён. Подключение

```
namespace my_namespace // объявляем пространство имён my_namespace
{
    int value = 10; // переменная value будет доступна в пространстве имён my_namespace
}
using namespace my_namespace; // подключаем пространство имён my_namespace
int main(int argc, char** argv)
{
    value = 5; // используем переменную value из пространства имён my_namespace без явного
указания
    return 0;
}
```

Напишем программу с подключением пространства имён

[Готовый пример кода]

Вложенные пространства имён

Ещё пространства имён могут быть **вложенными** — это когда мы объявляем одно пространство имён **внутри** другого пространства имён. Можно вкладывать друг в друга столько пространств имён, сколько пожелаете.

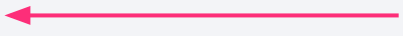
Это делается для лучшего структурирования кода.

Чтобы воспользоваться функциями или переменными из вложенного пространства имён, нужно полностью указывать весь путь, то есть все родительские пространства имён

Вложенные пространства имён

```
namespace my_namespace // объявляем пространство имён my_namespace
{
    namespace other_namespace // объявляем пространство имён other_namespace внутри
my_namespace
    {
        int value = 7;
    }
    int value = 10; // переменная value будет доступна в пространстве имён my_namespace
}

int main(int argc, char** argv)
{
    std::cout << my_namespace::value << std::endl; // 10
    std::cout << my_namespace::other_namespace::value << std::endl; // 7
    return 0;
} //  ???
```



Сколько пространств имён используется в этом коде?

Вложенные пространства имён

```
namespace my_namespace // объявляем пространство имён my_namespace
{
    namespace other_namespace // объявляем пространство имён other_namespace внутри
my_namespace
    {
        int value = 7;
    }
    int value = 10; // переменная value будет доступна в пространстве имён my_namespace
}

int main(int argc, char** argv)
{
    std::cout << my_namespace::value << std::endl; // 10
    std::cout << my_namespace::other_namespace::value << std::endl; // 7
    return 0;
} // 3: my_namespace, other_namespace, std
```

Итоги

Итоги

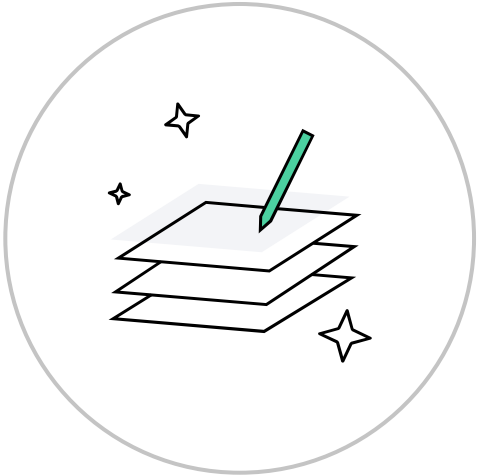
Сегодня мы:

1. Разобрались с типами памяти в C++
2. Посмотрели, что такое ключевое слово static
3. Узнали, что такое пространства имён
4. Выяснили, как пользоваться ключевым словом namespace

Домашнее задание

Давайте посмотрим ваше домашнее задание:

1. Вопросы по домашней работе задавайте в чате группы
2. Задачи можно сдавать по частям
3. Зачёт по домашней работе ставят после того, как приняты все задачи



Дополнительные материалы

- Типы памяти

Задавайте вопросы и пишите отзыв о лекции