Классы хранения

Лекция 7

Классы хранения

- Класс хранения в контексте объявления переменных C++ — это описатель типа, который управляет временем существования, компоновкой и расположением объектов в памяти.
- Каждый объект может иметь только один класс хранения.

Область видимости

- Блок – переменная видна от момента объявления до конца блока. Все локальные переменные (в том числе параметры функции) имеют блочную область видимости
- Функция – эта область видимости относится только к меткам. Метки видны во всей функции, где объявлены (в том числе и выше объявления)
- Файл – эту область видимости имеют все переменные, объявленные вне функций и сами функции. Область видимости начинается в момент объявления и длится до конца файла
- Класс – все поля классов имеют эту область видимости. Поля класса видны в любой функции класса.
- Глобальная – вся программа

Компоновка (связывание)

- Внешняя – переменная или функция видна вне файла, где объявлена
- Внутренняя – переменная или функция видна только внутри файла, где объявлена
- Без компоновки – блочные переменные не имеют компоновки

Длительность хранения

- Автоматическая –переменная начинает существовать с момента объявления и до конца блока, в котором объявлена
- Статическая – переменная начинается существовать с момента запуска программы и заканчивает существование в момент завершения программы

Спецификаторы классов памяти

- **`auto`** (устарело и в C++ 11 не используется как спецификатор класса памяти)
- **`register`** (в C++ 11 признано устаревшим)
- **`static`**
- **`extern`**
- **`thread_local`**

auto

- Блочная область видимости
- Отсутствие компоновки
- Автоматическая длительность хранения

Все локальные переменные и параметры функций по умолчанию относятся к этому классу памяти

В стандарте C++11 слово **auto** не используется для обозначения класса памяти

auto

Можно объявить несколько переменных с одинаковым именем в разных областях видимости. Если области видимости вложенные, то действует правило экранирования – внутренняя переменная перекрывает доступ к внешней

```
int main()
{
    int n = 10;
    {
        int n = 5;
        std::cout << n << std::endl;
    }
    std::cout << n << std::endl;
    return 0;
}
```

Результат:

```
5
10
```

register

- Блочная область видимости
- Отсутствие компоновки
- Автоматическая длительность хранения

Является рекомендацией компилятору разместить переменную в регистрах процессора.

В настоящий момент признано устаревшим:

- компиляторы хорошо оптимизируют код и сами помещают нужные переменные в регистры процессора
- может не оказаться свободных регистров процессора и рекомендация будет проигнорирована

static

Может применяться к:

- локальным переменным
- внешним переменным
- функциям

`static` с локальными переменными

- Блочная область видимости
- Внутренняя компоновка
- Статическая длительность хранения

Локальная переменная, объявленная со словом **`static`**:

- создается при первом входе в блок, где она объявлена
- «живет» до конца работы программы
- сохраняет свое последнее значение

static с локальными переменными

```
int counter()
{
    static int count; //по умолчанию count = 0
    return ++count;
}
```

```
int main()
{
    for(int i=0;i<10;i++)
        std::cout << counter() << std::endl;
    return 0;
}
```

static с локальными переменными

```
int & fun1(int a)
{
    int n = 0;
    n += a;
    return n;
}
```

Плохо! Возвращается ссылка, копия переменной n не создается, при выходе из функции n уничтожается, поведение программы не определено

```
int & fun1(int a)
{
    static int n = 0;
    n += a;
    return n;
}
```

Хорошо! Возвращается ссылка на статическую переменную, которая продолжает свое существование даже при выходе из функции

`static` с внешними переменными

- Файловая область видимости
- Внутренняя компоновка
- Статическая длительность хранения

Внешняя переменная, объявленная со словом **`static`**, видна только в том файле, где объявлена

Использование слова **`static`** с внешними переменными позволяет скрывать данные внутри модуля в многофайловых программах

`static` с функциями

Функция, объявленная со словом **`static`**, видна только в том файле, где объявлена

extern

Может применяться к:

- переменным
- функциям

По умолчанию все функции считаются объявленными со словом extern

extern с переменными

- Глобальная область видимости
- Внешняя компоновка
- Статическая длительность хранения

Используется как «чистое объявление». Сообщение компоновщику, что переменная объявлена в другом файле, память под нее при этом не выделяется

file1.cpp

```
int n = 10;
```

file2.cpp

```
int main()
{
    extern int n;
    {
        std::cout << n << std::endl;
    }
return 0;
}
```

thread_local

- Переменная, объявленная с описателем **thread_local**, доступна только в том потоке, в котором она была создана.
- Переменная создается при создании потока и уничтожается при его уничтожении.
- Описатель **thread_local** можно совместно использовать с **static** или **extern**.
- **thread_local** можно применять только в объявлениях и определениях данных. **thread_local** нельзя использовать в объявлениях и определениях функций.

Конец