Классы памяти. Препроцессор

Алтайский государственный университет
Факультет математики и ИТ
Кафедра информатики

Барнаул 2014

Лекция 9

- План
  - Классы памяти переменных
  - Препроцессор

Классы памяти переменных

- Автоматический класс
- Статический класс
- Регистровый класс
- Внешний класс
- Внешний статический класс
- Изменяемость переменных
- Общая схема описания переменных

Классы памяти переменных

- Класс памяти переменной определяет область видимости и время жизни переменной

- В языке Си переменные могут иметь один из следующих классов памяти:
  - **`auto`** – автоматический
  - **`static`** – статический
  - **`extern`** – внешний
  - **`register`** – регистровый
  - внешний статический

Классы памяти переменных

- Автоматический класс памяти (**auto**)
  - задается необязательным ключевым словом **auto** при описании переменной перед указанием типа
    - `auto float f=0;`
  - время существования переменной определяется областью видимости
    - переменная создается в начале блока {…}
    - переменная удаляется в конце блока {…}
  - при инициализации переменных допускается употребление выражений, включающих переменные и вызовы функций
    - `auto float a=10, b=2*a*sqrt(a);`

Классы памяти переменных

- Автоматический класс памяти (**auto**)
  - задается необязательным ключевым словом **auto** при описании переменной перед указанием типа
  - область действия существования переменной определяется областью видимости

```
void main() {
  auto int i;
  for (i=0; i<5; ++i) {
    int k=100;
    if(i%2) {
      int d = 3;
      k-=d;
    }
    k++;
  }
}
```

Область видимости i

Создание i в памяти

Инициализация i

Использование i

Удаление i

Классы памяти переменных

- Автоматический класс памяти (**auto**)
  - задается необязательным ключевым словом **auto** при описании переменной перед указанием типа
  - время существования переменной определяется областью видимости

```
void main() {
   auto int i;
   for (i=0; i<5; ++i) {
     int k=100;
     if(i%2) {
       int d = 3;
       k-=d;
     }
     k++;
   }
}
```

Область видимости k

Создание k в памяти

Инициализация k

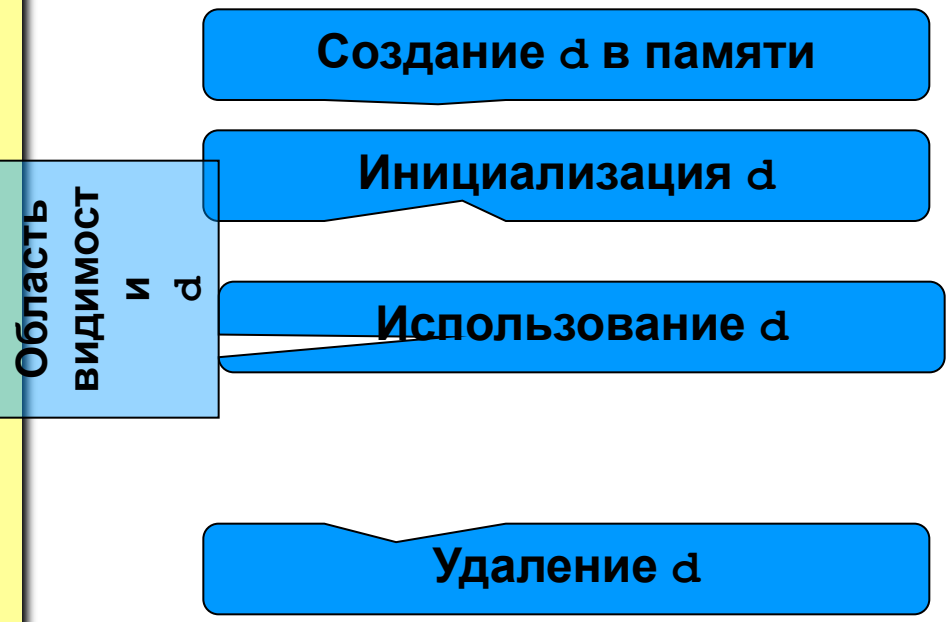
Использование k

Удаление k

Классы памяти переменных

- Автоматический класс памяти (**auto**)
  - задается необязательным ключевым словом **auto** при описании переменной перед указанием типа
  - время существования переменной определяется областью видимости

```
void main() {
  auto int i;
  for (i=0; i<5; ++i) {
    int k=100;
    if(i%2) {
      int d = 3;
      k-=d;
    }
    k++;
  }
}
```



Классы памяти переменных

- Автоматический класс памяти **auto**

```
#include <stdio.h>

void func(int p) {
 auto int a = 2;  /* автоматическая переменная */
 int b = sqrt(a); /* автоматическая переменная */
 b += 2*a + p;
 printf("a = %d, b = %d\n", a, b);
}

void main() {
  int i;           /* автоматическая переменная */
  for (i=0; i<5; ++i)
    if(i%2) {
      int d = 13; /* автоматическая переменная */
      func(i+d);
    }
}
```

Создание a,b в памяти

Инициализация a,b

Использование a,b

Удаление a,b

Классы памяти переменных

- Статический класс памяти (**`static`**)
  - задается ключевым словом **`static`** при описании переменной перед указанием типа
    - **`static int a=0;`**
  - область видимости переменной – блок `{…}`, в котором она определена
  - время существования переменной – сеанс работы программы
    - переменная создается **при старте программы**
    - переменная удаляется **при завершении программы**
  - инициализация переменных осуществляется **только при первом** входе в блок

```
static int a=0; /* инициализация однократна,        */
                /* между входами в блок             */
                /* значение переменной сохраняется */
```

  - параметры функций не могут быть статическими

Классы памяти переменных

- Пример. Автоматическая и статическая переменные

```
#include <stdio.h>
void stat(); /* прототип функ

void main() {
 int i;
 for (i=0;i<5;++i) stat();
}

void stat() {
  int a =          /* автоматиче       ерем
      0;
  static int s =   /* статическая переменная */
      0;
 printf("auto = %d, static = %d\n", a, s);
 ++a; ++s;
}
```

s создается при старте программы

s удаляется при завершении программы

s инициализируется только при первом вызове функции, сохраняя значение между вызовами

a создается и инициализируется при каждом вызове функции

Функции в Си

- Пример. Автоматическая и статическая переменные

```
auto = 0, static = 0
auto = 0, static = 1
auto = 0, static = 2
auto = 0, static = 3
auto = 0, static = 4
```

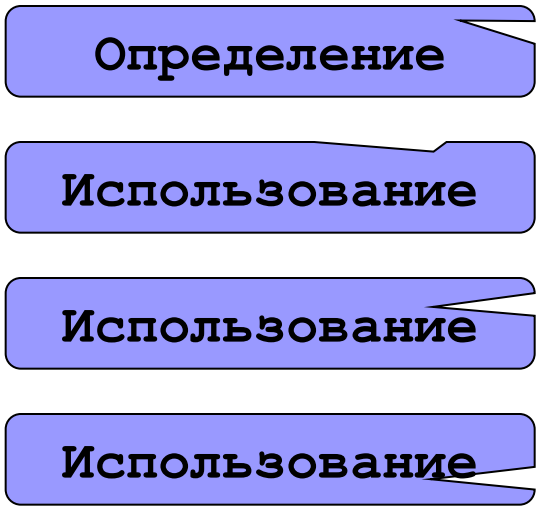
Классы памяти переменных

- Регистровый класс памяти (**register**)
  - задается ключевым словом **register** при описании переменной перед указанием типа
    - `register int k=0;`
  - при возможности регистровые переменные размещаются в регистрах процессора, а не в памяти
  - регистровые переменные не имеют адреса, т.е. к ним не применим оператор **&**
  - в остальном аналогичны автоматическим переменным
  - чаще всего регистровый класс памяти используется для переменных-счетчиков цикла

```
void main() {
  register int i;    /* регистровая переменная */
  for (i=0; i<15; ++i) printf("%d\n", i);
}
```

Классы памяти переменных

- Внешний класс памяти (**extern**)
  - внешние переменные – переменные, определенные вне функций
  - область действия внешних переменных – вся программа, т.е. внешние переменные глобальны



first.c

```
#include <stdio.h>

int Count=0;

int add(int a, int b) {
  Count++;
  return a+b;
}

int sub(int a, int b) {
  Count++;
  return a-b;
}

void main() {
  int x=1, y=10, z=0;
  z = add(x,y) + sub(y,x);
  printf("%d (%d)", z, Count);
}
```

Классы памяти переменных

- Внешний класс памяти (**extern**)
  - определение внешней переменной должно быть единственным на программу (только в одном файле)
    - `int global=100;`
    - при определении внешней переменной не требуется указывать специальных ключевых слов!
  - для использования внешних переменных, определенных за пределами данного файла, нужно поместить объявление внешней переменной
    - `extern int global;`

first.c

```
#include <stdio.h>

int Count=0;

int add(int a, int b) {
  Count++;
  return a+b;
}

int sub(int a, int b) {
  Count++;
  return a-b;
}

void main() {
  int x=1, y=10, z=0;
  z = add(x,y) + sub(y,x);
  printf("%d (%d)", z, Count);
}
```

Определение

Классы памяти переменных

- Внешний класс памяти (**extern**)
  - определение внешней переменной – в одном файле
  - в остальных – объявление переменной

Объявление

second.c

```
extern int Count;

int mul(int a, int b) {
  Count++;
  return a+b;
}
```

Объявление

third.c

```
extern int Count;

int div(int a, int b) {
  Count++;
  return a+b;
}
```

first.c

```
#include <stdio.h>

int Count=0;

int add(int a, int b) {
  Count++;
  return a+b;
}

int sub(int a, int b) {
  Count++;
  return a-b;
}

void main() {
  int x=1, y=10, z=0;
  z = add(x,y) + sub(y,x);
  printf("%d (%d)", z, Count);
}
```

Определение

Классы памяти переменных

- Внешний класс памяти (**extern**)
  - инициализировать внешние переменные можно только в определении (не в объявлении)
    - `int global=1024;` `/* корректно */`
    - `extern int global=0;` `/* некорректно */`
  - инициализировать внешние переменные можно только константными выражениями без вызовов функций
    - `int global=(8*1024)/2;` `/* корректно */`
    - `float wrong=2*sqrt(global);` `/* некорректно */`

Классы памяти переменных

- Внешний статический класс памяти
  - Внешние переменные могут быть объявлены как статические
  - Область видимости внешней статической переменной – файл, в котором она определена (а не вся программа)

Внешняя статическая переменная

Доступна в том же файле ниже определения

Недоступна в других файлах

first.c

```
#include <stdio.h>

static int Count=0;

int add(int a, int b) {
  Count++;
  return a+b;
}

void main() {
  int x=1, y=10, z=0;
  z = add(x,y);
  printf("%d (%d)", z, Count);
}
```

second.c

```
extern int Count; /* Ошибка! */

int mul(int a, int b) {
  Count++;
  return a+b;
}
```

Квалификаторы `const` и `volatile`

- Определение любой переменной может предваряться квалификаторами `const` или `volatile`
- Квалификатор `const` запрещает любые изменения значения переменной после ее инициализации
  - `const double PI = 3.141592;`
  - `const int N = 1000;`
- Квалификатор `volatile` извещает компилятор о том, что значение переменной может изменяться внешними по отношению к программе процессами
  - `volatile unsigned Time;`
  - учитывается при оптимизации кода программы

Описание переменных: общая схема

- Квалификаторы и модификаторы

Класс памяти	Изменяемость	Знаковость	Длина	Тип
auto	const	signed	short	int
static	volatile	unsigned	long	char
register			long long	float
extern				double

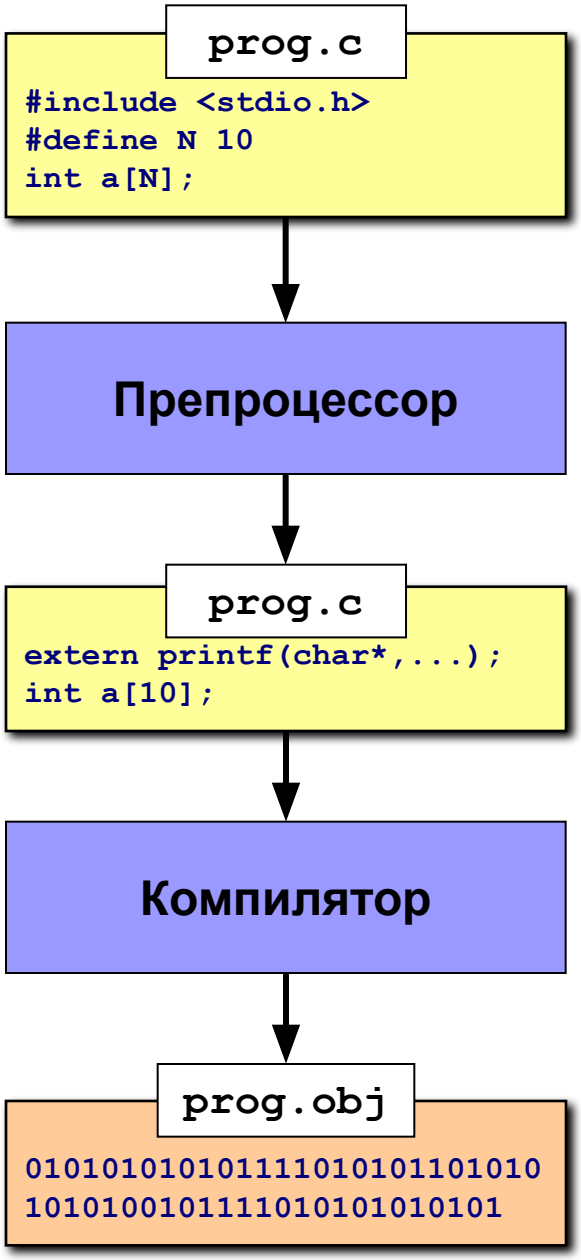
- Примеры:
  - `static volatile unsigned long long int TimeTicks;`
  - ~~`register const unsigned long double MyRealVar;`~~
  - `auto signed short int x_coord;`

Препроцессор

- Препроцессор: что это?
- Директивы препроцессора
- Подключение файлов
- Условная компиляция
- Макросы

Препроцессор: что это?

- Препроцессор – специальная программа, автоматически вызываемая компилятором перед собственно компиляцией
  - 1й проход: вызов препроцессора для Си-файла
  - 2й проход: вызов компилятора для измененного Си-файла



Директивы препроцессора

- Три основных типа директив
  - Подключение файла
    - `#include`
  - Условной компиляции
    - `#if, #ifdef, #ifndef, #else, #elif, #endif`
  - Макро-подстановка (макрос)
    - `#define, #undef`
- Правила построения директив
  - Всегда начинается с #
  - Может появляться в любом месте программы
  - В той же строке может содержаться комментарий
  - Воспринимается как одна строка, если явно не продолжена
    - `#define MAX_CHARS 300 /*Макс. длина имени файла*/`
    - ```
      #define MAX_FILES \
                      100
      ```
      ```

Подключение файлов

- Зачем?
  - Позволяет разделять программу или модули на интерфейс и реализацию *(см. Принципы структурного программирования)*
- Интерфейс (заголовочный файл) содержит все объявления модуля (константы, переменные, типы данных, функции)
  - Рекомендуемое расширение заголовочного файла: **`.h`**
- Имена заголовочных файлов пользователя – в “ … ”
  - **`#include “mydefs.h”`**
- Имена системных заголовочных файлов – в < … >
  - **`#include <stdio.h>`**

Условная компиляция

- Зачем?
  - Один и тот же исходный код для различных платформ
  - Необходимость иметь различный код для специфических ситуаций
- Условная компиляция
  - **`#ifdef name`**
  - **`#ifndef name`**
  - **`#if expr`**
  - **`#elif expr`**
  - **`#else`**
  - **`#endif`**
- **Удаление макро-определений**
  - **`#undef PLUSONE`**

```
#ifndef FOO_H
#define FOO_H

#ifdef WINDOWS_OS
#include <windows.h>
#elif LINUX_OS
#include <linux.h>
#endif
.
.
.
#endif
```

```
gcc -DWINDOWS_OS foo.c
```

Условная компиляция

- Другой пример

```
...
    if (some expr) {
        some code
#ifdef DEBUG
        printf("this path taken\n" );
#endif
    }
    else
        some other code;
```

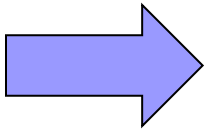
Макросы

- Предоставляют возможность параметризованной автоматической текстовой замены
- Зачем?
  - Порождаемый код иногда может быть быстрее
  - Нет контроля типов
- Макро-определение
  - `#define MAXLINE 120`
  - `#define lower(c) ((c)-`A'+`a')`
- Макро-подстановка
  - `char buf[MAXLINE+1];`
    превращается в
    `char buf[120+1];`
  - `c = lower(buf[i]);`
    превращается в
    `c = ((buf[i])-`A'+`a');`

Макросы: используйте ( )

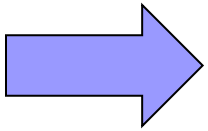
- Всегда заключайте параметры макро-функций в скобки!!!

```
#define plusone(x) x+1
…
i = 3*plusone(2);
…
```



```
…
i = 3*2+1;
…
```

```
#define plusone(x)
((x)+1)
…
i = 3*plusone(2);
…
```

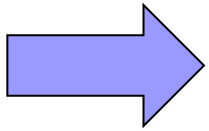


```
…
i = 3*((2)+1);
…
```

Макросы: думайте о побочных эффектах

- Частой причиной побочных эффектов являются операции “++“ и “--“
- Всегда избегайте дополнительных трюков с параметрами макро-функций

```
#define max(a, b) ((a)>(b)?(a):(b))
...
y = max(i++, j++);
...
```



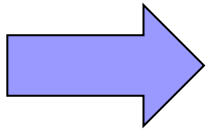
```
...
y = ((i++)>(j++)?(i++):(j++));
...
```

Данные какого типа можно передавать в `max()`?

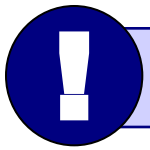
Макросы: оператор

- Оператор # в макросах конвертирует аргумент в строковую константу
- Всегда избегайте дополнительных трюков с параметрами макро-функций

```
#define PRINT_INT(x) printf( #x "= %d\n",
x)
...
PRINT_INT( x * y );
...
```



```
...
printf( "x * y" "= %d\n", x*y);
...
```

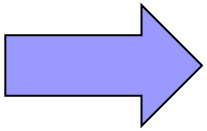


"abc" "defgh" для компилятора есть "abcdefgh"

Макросы: оператор

- Нужен крайне редко
- Склеивает две лексемы

```
#define GENERIC_MAX(type) \
type type##_max(type x, type y) \
{ return x > y ? x : y };
...
GENERIC_MAX(float)
...
```



```
...
float float_max(float x, float y)
{ return x > y ? x : y };
...
```

Другие директивы: #error

- Позволяет препроцессору инициировать ошибки компиляции
  - `#error сообщение`
- Пример

```
#if defined(WINDOWS)
...
#elif defined(LINUX)
...
#elif defined(MAC_OS_X)
...
#else
#error no OS specified
#endif
```

Другие директивы: #line

- Позволяет переопределить номер строки для компилятора
  - **#line n**

  или
  - **#line n “file”**
- Пример

one.c

```
main() {
#line 101 “two.c”
    i++;
}
```

```
> gcc one.c
> two.c: In function 'main':
> two.c:101: 'i' undeclared (first use in this function)
```

Макросы: некоторые общие свойства

- Макросы могут содержать макросы
  - Препроцессор может делать несколько проходов для повторной замены
- Макро-определение имеет силу до конца файла
- Макрос не может определяться дважды
- Перед повторным определением необходимо аннулировать предыдущее с помощью `#undef`

Препроцессор: резюме

- Препроцессор позволяет программисту автоматически модифицировать исходный код программы перед компиляцией
  - Подключение файлов
  - Условная компиляция
  - Макросы
- Макросы иногда полезны, но требуют повышенного внимания (опасны!)
  - Убедитесь, что Вы помните основные правила
  - Используйте скобки, где только можно
  - Думайте о побочных эффектах

Вопросы?

- Классы памяти переменных
  - Автоматический класс
  - Статический класс
  - Регистровый класс
  - Внешний класс
  - Внешний статический класс
  - Изменяемость переменных
  - Общая схема описания переменных
- Препроцессор
  - Препроцессор: что это?
  - Директивы препроцессора
  - Подключение файлов
  - Условная компиляция
  - Макросы

Т. Зеленченко Иллюстрация к произведению И.Ильфа и Е.Петрова "12 стульев"