ФУНКЦИИ В ЯЗЫКЕ СИ

лекция 5

План лекции

- Понятие подпрограммы и функции
- Параметры функции
- Возвращаемое значение функции
- Переменное число принимаемых параметров
- Время жизни и область видимости переменных
- Рекурсия

Понятие функции

- Подпрограмма (ПП) – это поименованный или иным образом идентифицированный фрагмент компьютерной программы, которому можно передать управление (*вызвать*) в любой её точке и который имеет возможность вернуть управление в точку, следующую за точкой своего вызова

- Maurice Wilkes, David Wheeler, and Stanley Gill "Preparation of Programs for an Electronic Digital Computer" 1951, 170с. цена 5.00USD

- Уменьшение размера памяти, занимаемой кодом программы – почти неактуально в наст. вр.
- Структуризация программы с целью удобства её понимания и сопровождения
  - Исправление ошибок, оптимизация, расширение функциональности в ПП автоматически отражается на всех её вызовах
  - Вынесение в ПП даже однократно выполняемого набора действий делает программу более понятной и обозримой

Понятие функции – механизм вызова

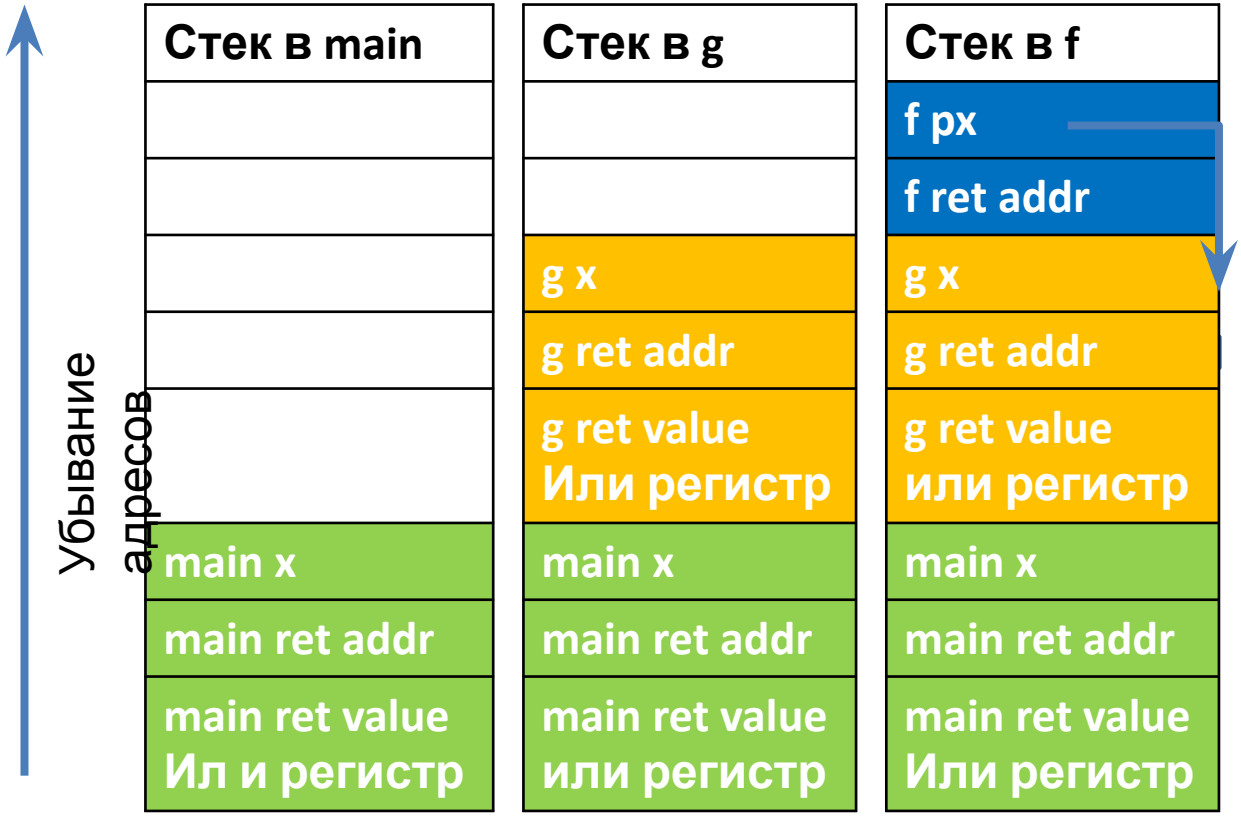
- Вызов ПП делится на
  - Подготовительные служебные действия вызывающей программы
  - Собственно работу ПП
  - Заключительные служебные действия вызывающей программы
- Каждому вызову ПП соответствует отдельная область памяти – *стековый кадр*
- Стековый кадр
  - Существует на протяжении всего вызова ПП
    - Включая вложенные вызовы других ПП
  - Перестаёт существовать после завершения вызова ПП
- Стековым кадром могут пользоваться
  - Программа
  - ПП
  - Другие ПП, вызванные из ПП

Понятие функции – механизм вызова

- Стековый кадр содержит
  - *Адрес возврата* – адрес команды, которая получит управление после завершения работы (*выхода*) из ПП
    - Вычисляется процессором или компилятором
    - Обычно адрес первой команды заключительных действий
  - *Параметры* ПП – переменные ПП, значения которых вызывающая программа устанавливает перед вызовом
    - Могут частично находиться в регистрах процессора
  - Внутренние переменные ПП
  - *Результат* работы ПП – ячейка памяти, значение которой устанавливается ПП перед выходом и после этого может использоваться вызывающей программой
    - Может находиться в регистре процессора

Примеры стековых кадров

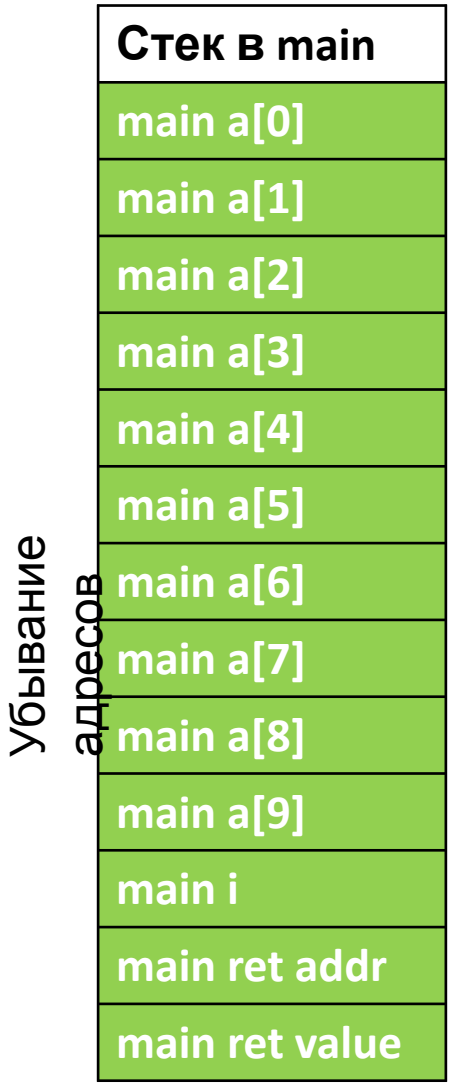
```
void f(int *px) { *px = 1; }
int g()
{
    int x;
    f(&x);
    return x;
}
int main ()
{
    int x = g();
    return 0;
}
```



Примеры стековых кадров

```
int main()
{
    int a[10],i;
    for (i=0;i<=10;i++)
        a[i]=0;
    return 0;
}
```

Что делает эта программа?



Понятие функции – описание, вызов

- В языке Си подпрограммы называются *функциями*
- Описание функции делится на
  - *Заголовок* -- тип результата, имя функции и список параметров функции
    - Если тип-результата есть void, то функция не возвращает результата -- аналог процедуры в языке Паскаль
  - *Тело* – это набор инструкций, который будет выполнен, когда функция будет вызвана
- Вызов – это частный случай постфиксного выражения языка Си, см. Лекцию 4

Понятие функции – описание

- Синтаксис описания функций, начиная с C89 и до наст. времени
  тип-результата имя-функции (список-типов-параметров)
  {
  объявления
  инструкции
  }
  - Описывает значение типа "функция, возвращающая тип-результата и имеющая параметры список-типов-параметров"
  - Если список-типов-параметров заканчивается лексемами ',' '…' (в Си) или '…' (в Си++), то функция имеет переменное число параметров
- Устаревший синтаксис, всё ещё разрешённый в C89, C99, C11
  тип-результата имя-функции (список-имён-параметров, если он есть)
  объявление-параметров;
  {
  объявления
  инструкции
  }
  - Описывает значение типа "функция, возвращающая тип-результата и имеющая неизвестное число параметров неизвестных типов"

Понятие функции – описание

<список-типов-параметров> ::=
    <список-параметров>
| <список-параметров> ',' '...'

<список-параметров> ::=
    <объявление-параметра>
| <список-параметров> ',' <объявление-параметра>

<объявление-параметра> ::=
    <спецификаторы-объявления> <объявитель>
| <спецификаторы-объявления> [<абстрактный-объявитель>]

Понятие функции – описание

- Работа функций типа void завершается
  - Исполнением инструкции return;
  - Исполнением последней инструкции тела функции

- Работа функций других типов завершается
  - Исполнением инструкции return выражение;
    - Значение выражения является результатом работы функции
    - Значение выражения будет преобразовано к типу результата функции с помощью неявных преобразований (см. Лекцию 4) или компиляция функции закончится ошибкой
  - Исполнением последней инструкции тела функции
    - Результат работы функции в этом случае неопределён – возможно мы получим сообщение об этом от компилятора

Понятие функции – вызов

- Вызов функции имеет вид
  постфиксное-выражение (список-аргументов-выражений)
- постфиксное-выражение
  - Объявленный и/или описанный идентификатор функции
  - Переменная типа указатель на функцию или выражение типа указатель на функцию
  - Ранее необъявленный идентификатор
    - Автоматически объявляет идентификатор как функцию, возвращающую int с неизвестным числом и типами параметров
    - Источник ошибок
- список-аргументов-выражений
  - Проверка соответствия числа аргументов-выражений и числа параметров функции
    - Если число параметров известно, то строгая проверка
    - Если переменное число параметров, то число аргументов-выражений >= число параметров
  - Проверка соответствия типов аргументов-выражений и типов параметров функции
    - Если типы известны, то строгая проверка и, возможно, неявное преобразование
    - Если типы неизвестны, то float -> double + целочисленное повышение

Понятие функции – описание

- void my_f() {} // Старый синтаксис
- void my_g(void) {} // Новый синтаксис
- int my_fact(int n) {return n==0 ? 1 : my_fact(n-1)*n;}
  - Что делает эта функция?
  - Чему равно my_fact(5)?
- int my_fib(int n)
  {return n<=1 ? 1 : my_fib(n-1)+my_fib(n-2);}
  - Чему равно my_fib(5)?

Понятие функции – описание

- int my_fact(int n) {return n==0 ? 1 : my_fact(n-1)*n;}
- int my_fib(int n)
  {return n<=1 ? 1 : my_fib(n-1)+my_fib(n-2);}
- int (*my_fun_factory(int n))(int)
  {return n == 0 ? my_fact : my_fib;}
  - Чему равно my_fun_factory(0)(5)?
  - Чему равно my_fun_factory(1)(5)?

- Понятие подпрограммы и функции
- Параметры функции
- Возвращаемое значение функции
- Переменное число принимаемых параметров
- Время жизни и область видимости переменных
- Рекурсия

Переменное число принимаемых параметров

- Описание функции с переменным числом параметров на языке Си имеет вид

  тип-результата имя-функции (список-типов-параметров, ...)
  {
  объявления
  инструкции
  }

- Язык Си++ разрешает пропускать ',' перед '…'
- Включено в стандарты языка Си C89, C99, C11
  - Существовали и до C89

Переменное число принимаемых параметров

- Запись вызова функции с переменным числом параметров не отличается от записи вызова обычной функции

  постфиксное-выражение (список-аргументов-выражений)

- Проверка соответствия числа аргументов-выражений и числа параметров функции
  - Число аргументов-выражений должно быть >= число параметров до лексемы '...'

- Проверка соответствия типов аргументов-выражений и типов параметров функции
  - До лексемы '...' строгая проверка и, возможно, неявное преобразование
  - После лексемы '...' преобразование float -> double и целочисленное повышение

Доступ к значениям параметров, переданных через '...'

- Заголовочный файл stdarg.h
  - C89 и далее
  - До C89 varargs.h

- va_list – значения параметров, переданные через '...'
- va_start – инициализация переменной типа va_list
- va_arg – извлечение значения очередного аргумента из значения переменной типа va_list *и переход к следующему аргументу*
- va_end – завершение работы с переменной типа va_list
- va_copy – копирование значения из одной переменной типа va_list в другую переменную типа va_list (C99 и C11)

Пример

```
#include <stdio.h>
#include <stdarg.h>
void print_int_args(int arg1, ...)
{
  va_list ap;
  int i;
  va_start(ap, arg1);
  for (i = arg1; i >= 0; i = va_arg(ap, int))
    printf("%d ", i);
  va_end(ap);
  printf("\n");
}
int main(void)
{
   print_int_args (-1); // что будет напечатано?
   print_int_args (5, 2, 14, 84, 97, 15, 24, 48, -1);
   print_int_args (84, 51, -1);
   // print_int_args (); -- синтаксическая ошибка или предупреждение
   // print_int_args (84, 51); -- ошибка времени исполнения
   print_int_args (0.5, -1); // 0.5 -> 0
   // print_int_args (0.5, 1.5, -1); -- ошибка врем. исполнения для 1.5
   return 0;
}
```

Ограничения на функции с переменным числом параметров

- Нельзя проверить, кончились ли значения параметров, переданные через '...'
  - Используем какой-либо явный признак конца списка значений
- Нельзя передать *все* значения параметров, полученные через '...', другой функции с переменным числом параметров
  - Каждой функции с переменным числом параметров – аналогичную функцию с постоянным числом параметров, последний из которых va_list
    - int printf(const char*f, ...)
    - int vprintf(const char *f, va_list vals)
- Нельзя проверить типы значений параметров, переданных через '...'

- Понятие подпрограммы и функции
- Параметры функции
- Возвращаемое значение функции
- Переменное число принимаемых параметров
- Время жизни и область видимости переменных
- Рекурсия

Область видимости переменных

- *Область видимости идентификатора* – часть программы, где использование этого идентификатора не вызывает ошибок компиляции и сборки
  - Вся программа (глобальная ОВ, она же внешняя ОВ)
  - Единица компиляции
  - Блок кода от '{' до соотв. '}'
- Это понятие относится ко компиляции и сборке программы

Область видимости переменных

- Вся программа (глобальная ОВ, она же внешняя ОВ)
  - extern и по умолчанию
- Единица компиляции
  - static вне всех блоков кода, ограниченных '{' и '}'
- Блок кода от '{' до соотв. '}'
  - По умолчанию
  - Вложенный блок кода может скрыть идентификатор

```
{
  int x;
  {
        int x[5];
  }
  if (sizeof(x) != sizeof(int)) printf("Так не может быть!\n");
}
```

Время жизни переменных

- *Время жизни переменной* – интервал времени, в течение которого для хранения значения переменной выделены ячейки памяти
  - Программа
  - Один блок кода от { до }
    - Например, всё тело функции
  - Поток многопоточной программы
- Это понятие относится ко времени исполнения программы

Время жизни переменных

- Программа
  - Все переменные, имеющие глобальную ОВ
  - Все переменные, объявленные как static
- Один блок кода от { до }
  - По умолчанию
- Поток многопоточной программы
  - C11 – квалификатор типа _Thread_local
    - _Thread_local int x;

Время жизни и область видимости переменных – пространства имён

- Каждый идентификатор попадает в одно из нескольких пространств имён
- Компилятор решает, из какого пространства имён взять идентификатор, по тексту вокруг идентификатора
  - Например, идентиф. после -> имеет смысл имени поля структуры
- Пространства имён в языке Си
  - Переменные, функции, typedef-имена и enum-константы
  - Метки инструкций
  - Идентификаторы, следующие за ключевыми словами struct, union, enum
  - Поля каждой структуры или объединения

Время жизни и область видимости переменных

- Как описать переменную, имеющую заданные область видимости и время жизни

		Время жизни переменной	
		Вся программа	Блок от { до }
Область видимости	Вся программа	(по умолчанию или extern) + вне всех { }	А смысл?!
	Вся единица компиляции	static + вне всех { }	А смысл?!
	Блок от { до }	(static или extern) + внутри { }	(по умолчанию или auto) + внутри { }

Время жизни и область видимости переменных

- Как понимать описание переменной

	Нет описания	**Описание без auto, extern, static**	**auto**	**static**	**extern**
вне { }	Вся программа, всё время работы int	Вся программа, всё время работы	Ошибка	Единица компиляции, всё время работы программы	Вся программа, всё время работы программы
внутри { }		Только блок, только время работы блока	Только блок, только время работы блока	Только блок, всё время работы программы	Только блок, всё время работы программы

Заключение

- Понятие подпрограммы и функции
- Параметры функции
- Возвращаемое значение функции
- Переменное число передаваемых параметров
- Время жизни и область видимости переменных
- Рекурсия