Программирование на языке C

Дискретный анализ 2012/13

Андрей Калинин, Татьяна Романова

3 сентября 2012 г.

Библиотеки

- libc
- Процесс компиляции
- Традиции языка C

Динамическая память

Профилирование

C++

- Особенности
- Недостатки

Литература

- Кернинган, Ричи: Язык программирования C.

Раздел

Скомпилируется ли эта программа?

```
main() {
   printf("Hello, world!\n");
}
```

Скомпилируется ли эта программа?

```
main() {
   printf("Hello, world!\n");
}
```

```
$ gcc t01.c
t01.c: In function 'main':
t01.c:2: warning: incompatible implicit declaration of
built-in function 'printf'
$ ./a.out
Hello, world!
```

Убираем предупреждение

```
int printf(const char *, ...);

main() {
   printf("Hello, world!\n");
}
```

```
$ gcc t02.c
$ ./a.out
Hello, world!
```

Что содержится в stdio.h?

```
#ifndef _STDIO_H_
#define _STDIO_H_
  ...
typedef struct __sFILE {
  unsigned char *_p; /*current position in buffer*/
  int     _r;     /* read space left for getc() */
  int     _w;     /* write space left for putc() */
  ...
}
  ...
int     printf(const char * __restrict, ...);
  ...
#endif /* _STDIO_H_ */
```

stdio.h

1. Определяются структуры, константы, макросы.
2. Нет определения функций, только объявления.
3. Включаются другие заголовочные файлы.
4. Защищающие директивы препроцессора (ifndef/define в начале, endif в конце файла)

Где находится определение функции printf?

stdio.h

1. Определяются структуры, константы, макросы.
2. Нет определения функций, только объявления.
3. Включаются другие заголовочные файлы.
4. Защищающие директивы препроцессора (ifndef/define в начале, endif в конце файла)

Где находится определение функции printf? В libc.a (обычно находящейся в каталоге /lib, /usr/lib или /usr/local/lib)

Что такое libc?

1. Библиотека стандартных функций языка C и основного интерфейса операционной системы Unix.
2. Подключается статически (libc.a) или динамически (libc.so).
3. libc.a: архив объектных файлов, созданный программой ar. Можно посмотреть содержимое командой ar t libc.a
4. Неявно подключается при создании выполнимой программы компилятором gcc.
5. libc.so: более сложная структура, похожая на выполнимый файл, реально подключается на этапе выполнения программы.

Содержимое libc.a

```
$ ar t /usr/lib/libc.a
wmemset.o
wmemmove.o
wmemcmp.o
...
printf.o
...
```

/usr/src/lib/libc/stdio/printf.c

```
#include <sys/cdefs.h>
#include <stdio.h>
#include <stdarg.h>

int printf(char const * __restrict fmt, ...)
{
        int ret;
        va_list ap;

        va_start(ap, fmt);
        ret = vfprintf(stdout, fmt, ap);
        va_end(ap);
        return (ret);
}
```

Раздел

Основные этапы сборки исполняемого файла

1. Обработка препроцессором, cpp: выполнение #-директив `#include`, `#define`, `#ifdef` и т.п.
2. Компиляция (выполняет и препроцессор): gcc -c file.c, получается объектный файл.
3. Редактирование связей: ld, объединяет все объектные файлы, как указанные явно, так и содержащиеся в библиотеках, в исполняемый файл.

Директивы препроцессора

- Выполняются для всех компилируемых файлов.
- Нужно следить, чтобы подключались одни и те же файлы: одинаковые настройки при компиляции, include-пути и прочее.
- Препроцессор — не компилятор! Множество подводных камней.
- Тем не менее, широко используется. К примеру, /usr/include/sys/queue.h, реализация очереди целиком на препроцессере.
- Популярно комментирование блоков при помощи препроцессора: `#if 0` ... `#endif`.
- Препроцессор может замедлять выполнение компиляции: pre-compiled заголовочные файлы.

Компиляция

- Компилятору нужно только описание вызова подпрограммы.
- Если такого нет, он сгенерирует описание неявно: int foo(...)
- При передаче указателей на структуры не нужно описание самих структур.

Редактор связей

- Производит поиск символьных названий и заменяет их на адреса настоящих переменных и функций.
- Общий для C, C++ и множества других языков: ld.
- Имеет собственные ограничения (например, на максимальное количество символов в названии функции или символ подчёркивания).
- В некоторых случаях важен порядок указания объектных файлов и библиотек.

Динамические библиотеки

- Используются неявно (можно отключить при помощи ключа -static)
- Реальное подключение происходит во время выполнения.
- DLL HELL: разные версии одной и той же библиотеки на разных компьютерах.
- Проверка версионности через названия файлов, /lib/libc.so.5.

Раздел

Модульное программирование

- Отсутствует в языке, но есть ряд соглашений по совместному использованию препроцессора и компоновщика.
- Интерфейс располагается в заголовочных h-файлах.
- Реализация скрыта в одном c-файлах.
- Для использования функций модуля в нескольких конфигурациях используются хендлы (объекты).
- ООП лишь более явная реализация идеи модульного программирования.

Примеры объектов в языке C

- Файлы UNIX: хендлы целые числа; представляют из себя сетевые соединения, устройства, файловую систему и т.п.
- stdio: FILE* — объект; fopen/popen/fcreate — конструкторы; fclose — деструктор; fprintf/fwrite/fread — методы.

Интерфейс словаря

```
#ifndef __DICT_H__
#define __DICT_H__

#ifdef __cplusplus
extern "C" {
#endif
   /* ... */
#ifdef __cplusplus
}
#endif

#endif /* __DICT_H__ */
```

Интерфейс словаря

```
typedef struct dict dict;

dict *dict_create();
dict *dict_load(const char *fname);
int   dict_add(dict *d, const char *key, int val);
int   dict_find(dict *d, const char *key, int *val);
int   dict_save(dict *d, const char *fname);
int   dict_close(dict *d);
```

Реализация словаря

```c
#include "dict.h"

struct dict {
  /* ... */
}

dict *dict_create() {
  dict *result = calloc(1, sizeof(dict));
  /* ... */
  return result;
}
```

Что не является раздельной компиляцией

Вынесение всех функций в заголовочный файл не позволит сделать библиотеку, т.к. ими можно будет пользоваться только в одной единице компиляции.

Раздел

Какая бывает память?

- Сегмент данных: статические переменные, выделяется на этапе компиляции.
- Стек: аргументы функций, адреса возврата, локальные переменные функций, выделяется на этапе времени выполнения. Время жизни определяется порядком выполнения функций. Доступен через alloca.
- Куча (динамическая память). Выделяется и уничтожается самостоятельно программистом.

Работа с кучей

- Попросить память: malloc.
- Изменить размер блока: realloc.
- Вернуть память: free.

Куча

- Куча может быть системной или библиотечной.
- Куча фрагментируется от частого получнеия и возвращения памяти.
- Библиотечная куча может не возвращать память системе.

Раздел

Профайлеры

- Наблюдение изнутри (gprof, gcov) и снаружи (shark) программы.
- Изнутри: теоретически точнее, но может влиять на ход выполнения.
- Снаружи: практически значимый результат, но иногда не хватает точности.

gprof

- gcc -pg
- Запустить программу.
- gprof имя-программы имя-проргаммы.gmon

gcov

- gcc -ftest-coverage -lgcov
- Запустить программу.
- gcov путь-к-исходнику

Раздел

ld ничего не знает о C++

- Шаблоны?
- Члены классов?
- Функции с одинаковыми названиями, но разными аргументами?
- Операторы?

Как это всё увязать с C-шным редактором связей?

Изменение имён функций

- C добавляет символ подчёркивания.
- C++ добавляет всю информацию о вызове функции в название. Для gcc:
  - int A::f(): _ZN1A1fEv
  - int A::f(int): _ZN1A1fEi
  - int A::f(int, int): _ZN1A1fEii
  - int A::f(double): _ZN1A1fEd
- c++filt.
- export "C"

Раздел

Проблемы

- Сложная грамматика: `AA BB(CC);` — объект или функция?
- STL: сложнее исходной задачи, при этом нет многих необходимых .
- Исключения: как освобождать ресурсы?
- Перегрузка операторов (operator« для вывода, operator/ для конкатенации строк)
- Шаблоны и inline-функции: всё в заголовчных файлах.
- Сложный стандарт: нет ни одного компилятора, ему соответствующего.
- Каждый C++-компилятор имеет свои ошибки и особенности.
- Диалекты языка промышленных компиляторов: Microsoft Visual C++, gcc.

Сообщения об ошибках

```
typedef std::map<std::string,std::string>
  StringToStringMap;
void print(const StringToStringMap& dict) {
  for(StringToStringMap::iterator p=dict.begin();
      p!=dict.end(); ++p) {
    std::cout << p->first << " -> "
      << p->second << std::endl;
  }
}
```

Текст ошибки

```
test.cpp: In function 'void print(const StringToStringMap&)':
test.cpp:8: error: conversion from
'std::_Rb_tree_const_iterator<std::pair<const
std::basic_string<char, std::char_traits<char>,
std::allocator<char> >, std::basic_string<char,
std::char_traits<char>, std::allocator<char> > > >'
to non-scalar type 'std::_Rb_tree_iterator<
std::pair<const std::basic_string<char,
std::char_traits<char>, std::allocator<char> >,
std::basic_string<char, std::char_traits<char>,
std::allocator<char> > > >' requested
```