Chapter 1

Вопросы

Вопрос 1 Программа, состоящая из нескольких файлов

- компиляция и линковка
- заголовочные файлы
- утилита make

Вопрос 2 Указатели, массивы, ссылки

- применение указателей и ссылок
- арифметика указателей

Вопрос 3 Три вида памяти. Работа с кучей на C

- глобальная/статическая память, стек, куча
- `malloc/calloc/realloc/free`
- `void*`

Вопрос 4 Структуры. Неинтрузивный связный список на C

- неинтрузивная реализация
- typedef

Вопрос 5 Структуры. Интрузивный связный список на C

- интрузивная реализация
- typedef

Вопрос 6 Функции. Указатели на функции

- как происходит вызов функции
- реализация сортировки
- `void sort(void* base, size_t num, size_t size, int (*compar)(const void*,const void*));`

Вопрос 7 Обзор стандартной библиотеки C

- `string.h (memcpy, memcmp, strcpy, strcmp, strcat, strstr, strchr, strtok)`
- `stdlib.h (atoi, strtoll, srand/rand, qsort)`

Вопрос 8 Ввод-вывод на C. Текстовые файлы

- FILE, fopen, fclose, r/w, t/b
- stdin, stdout, stderr
- printf, scanf, fprintf, fscanf, sprintf, sscanf, fgets
- обработка ошибок, feof, ferror

Вопрос 9 Ввод/вывод на C. Бинарные файлы

- FILE, fopen, fclose, r/w, t/b, буферизация
- fread, fwrite, fseek, ftell, fflush
- обработка ошибок, feof, ferror

Вопрос 10 Классы и объекты

- инкапсуляция: private/public
- конструктор (overloading), деструктор
- инициализация полей (в том числе C++11)
- C++11: =default, constructor chaining

Вопрос 11 Работа с кучей на C++

- new/delete
- создание объектов в куче
- конструктор копий
- оператор присваивания
- C++11: =delete

Вопрос 12 Наследование и полиморфизм

- protected
- virtual (overriding)
- таблица виртуальных функций
- статическое/динамическое связывание

Вопрос 13 Умные указатели

- `scoped_ptr`
- `unique_ptr`
- `shared_ptr`

Вопрос 14 Перегрузка операторов

- бинарные и унарные
- в классе/вне классе
- приведение типов

Вопрос 15 Ключевые слова extern, static, inline

- extern у переменных
- static у переменных и функций
- static у полей и методов
- inline у функций

Вопрос 16 Разное

- ключевое слово const (C/C++)
- перегрузка функций
- параметры функций по умолчанию

Вопрос 17 Наследование: детали

- сортировка и структуры данных C vs ООП
- private/protected наследование
- C+11: final, override

Вопрос 18 Элементы проектирования

- декомпозиция программы (Model, View)
- автотесты

Вопрос 19 Множественное наследование

- разрешение конфликтов имен
- виртуальное наследование
- наследование интерфейсов

Chapter 2

Ответы

Вопрос 1 Программа, состоящая из нескольких файлов

- компиляция и линковка

Listing 2.1: main.c

```
int main() {
    return 0;
}
```

Причины разбиения на файлы:

1. Абстракция
2. Несколько программистов
3. Быстродействие

Пусть в программе будет несколько файлов:

Listing 2.2: main.c

```
int main() {
    hello();
    return 0;
}
```

Listing 2.3: hello.c

```
void hello() {
    printf("Hello!");
}
```

Для компиляции:

```
$ gcc main.c hello.c -o main
```

Что происходит во время выполнения команды? Каждый файл компилируется по отдельности. В памяти каждый блок соответствует функции.

1. компиляция: из $*.c$ получаются объектные файлы $*.o$
2. линковка: выполняется линковщиком, задача состоит в том, чтобы собрать в один исполняемый файл, объединить блоки в памяти межу собой и выполнить разрешение адресов. Линковщик устанавливает относительные адреса, линковка более быстрый процесс.

   Если мы запустим компиляцию от объектных файлов, будет выполняться только линковка.

Пусть теперь сигнатура *hello* поменялась: теперь там есть параметры. Если скомпилируем по отдельности проблем не будет, но ошибка останется не замеченной. Для этого используются заголовочные файлы.

- заголовочные файлы У функции есть

1. Определение (definition)
2. Объявление (declaration) : сигнатура - получаемое и возвращаемое

Listing 2.4: hello.h

```
void hello(int n);
```

Listing 2.5: hello.c

```
#include "hello.h"
void hello(int n) {
    printf("%d", n);
}
```

Listing 2.6: main.c

```
#include "hello.h"
int main() {
    hello();
    return 0;
}
```

Теперь одно определение подключается в оба файла. Можно не указывать имя переменной в определении.

Если мы хотим ссылаться в цикле

Listing 2.7: a.h

```
#ifndef _a_H_
#define _a_H_
#include "b.h"
#endif
```

Listing 2.8: b.h

```
#ifndef _b_H_
#define _b_H_
#include "a.h"
#endif
```

- утилита make
  GCC:
  - только препроцессор → все #...

    ```
    $ gcc -E
    ```

  - только компиляция

    ```
    $ gcc -c
    ```

  - только перевод в ассемблер

    ```
    $ gcc -s
    ```

  Для автоматизации используется *make*.

Listing 2.9: Makefile

```
main: main.o str.o util.o
    gcc  main.o str.o util.o -o main
main.o: main.cpp util.h str.h
    gcc -c main.cpp
str.o: str.cpp str.h
    gcc -c str.cpp
util.o: util.cpp util.h
    gcc -c util.cpp
```

```
9
10 clean:
11 	rm -rf *.o
```

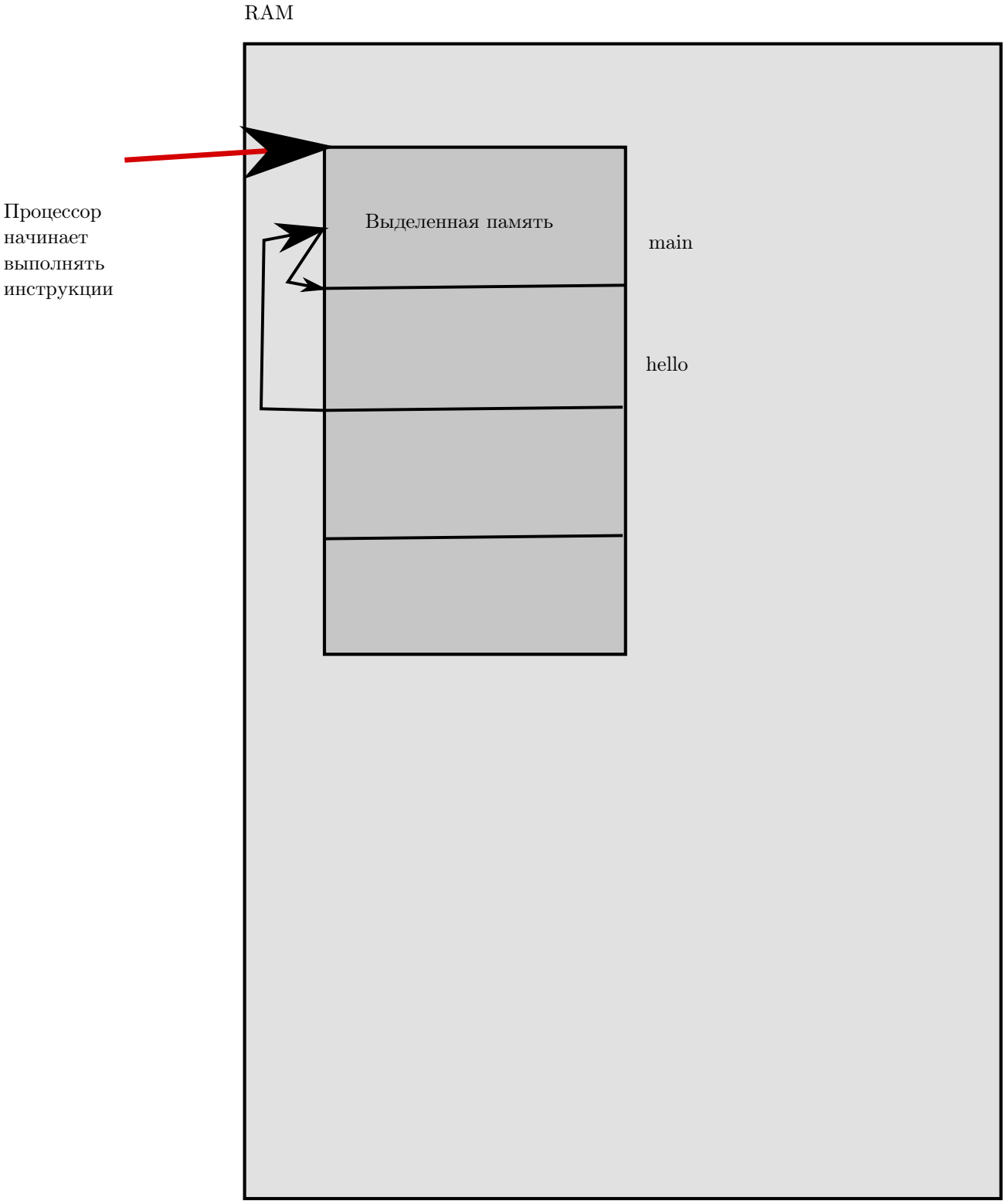

Figure 2.1: ram