Основы языка C, простейшие программы

А. Г. Фенстер, fenster@fenster.name

12 февраля 2010 г.

Конспект семинара №1 по программированию для студентов 1 курса ММФ и ФИТ НГУ. Помните, что чтение конспекта не делает посещение соответствующего семинара необязательным! О найденных ошибках и неточностях, пожалуйста, пишите мне по адресу fenster@fenster.name. Также буду рад получить любые комментарии по поводу этого текста.

1 Введение

Программы, исполняемые компьютерами под управлением современных операционных систем (Linux, Windows, ...), обычно должны быть представлены в некотором специальном виде, понятном операционной системе (и обычно нечитаемом для человека — попробуйте открыть в текстовом редакторе любой exe-файл). Программист, в свою очередь, хочет писать программы на языке, как можно более близком к естественному (человеческому) языку.

Языки программирования высокого уровня являются некоторым компромиссом: программист может не думать об устройстве и командах процессора и описывать задачу в понятном как человеку, так и компьютеру виде.

Текст, написанный на таком языке, далее может быть либо *интерпретирован* программой-*интерпретатором* — т. е. каждая инструкция будет прочитана и сразу же исполнена — либо *откомпилирован* (*оттранслирован*) в программу на некотором другом (вообще говоря — любом) языке, чаще всего — в исполняемый файл в формате, понятном данной операционной системе; этим занимаются программы-*компиляторы*.

В нашем курсе мы будем изучать язык программирования C («си» — третья буква английского алфавита). Программы на языке C обычно

не интерпретируются «на лету», а компилируются в исполняемый код. Компиляторов с языка C много и для решения наших учебных задач подойдёт практически любой из них. На практических занятиях будет показано, как использовать компилятор `gcc` в ОС Linux, на компьютерах в терминальном классе также установлена среда Microsoft Visual Studio, содержащая в том числе и компилятор с языка C.

Независимо от того, какой компилятор вы используете, работа с ним обычно проходит по следующему сценарию:

1. Набор текста программы на языке C в редакторе.
2. Компиляция программы в исполняемый файл.
3. Запуск полученного исполняемого файла.

При работе в командной строке ОС Linux все три действия выполняются по отдельности (см. команды Linux), при использовании *интегрированной среды* типа Microsoft Visual Studio эти три действия выполняются прямо в ней.

2 Основы языка C

Нет никакого смысла подробно описывать здесь действие каждой инструкции языка. За таким описанием обратитесь, например, к книге Кернигана и Ритчи «Язык C» (есть в библиотеке). Ниже в виде «шпаргалки» будут перечислены вещи, которые понадобятся нам на первых занятиях.

Как и в большинстве других языков, переменные и функции могут иметь имена, состоящие из букв, цифр и знака подчёркивания, не начинающиеся с цифры и не совпадающие ни с одним из *зарезервированных слов* языка.

Каждая переменная имеет *тип*, который определяет множество возможных значений переменной, допустимые операции над ней и размер, который переменная занимает в памяти. Основной тип — `int`; переменные этого типа могут хранить целое число со знаком. В большинстве современных реализаций переменная типа `int` занимает 4 байта. Помимо `int` нам в ближайшее время понадобятся типы `double` (вещественное число) и `char` (символ; на самом деле тоже число, но занимающее ровно 1 байт). Существуют также «беззнаковые» варианты целочисленных типов: `unsigned int` и `unsigned char`, они хранят натуральные числа.

Описание функции:

```
тип_результата имя(список_аргументов)
{
        ...
        return результат;
}
```

Инструкция `return` указывает результат функции и прекращает её дальнейшее выполнение.

Подключение «заголовочных файлов»

```
#include <stdio.h> /* ввод/вывод */
#include <math.h>  /* матем. функции */
```

Заголовочные файлы содержат описания типа результата и аргументов функций. Чтобы использовать стандартную функцию, необходимо «включить» (`#include`) заголовочный файл, содержащий информацию о ней.

Вывод на экран:

```
   printf("строка_форматирования", ...);
   printf("This is a text\n");
   printf("Числа: %d, %f\n", 7, 1.2);
```

Пусть объявлены переменные:

```
   int a = 7; double b = 1.2;
```

тогда их значения можно вывести так:

```
   printf("значения: %d, %f\n", a, b);
```

`%d` ↔ `int`, `%f` ↔ `double`, `%c` ↔ `char`

Пример «Hello World!»:

```
#include <stdio.h>

int main()
{
        printf("Hello World!\n");
        return 0;
}
```

Логические операторы

0 — «ложь», остальные числа — «истина»

`&&` — логическое «и»: оба условия выполнены

```
        (5 > 3) && (7 == 7) — истина
        (5 < 3) && (7 == 7) — ложь
```

`||` — логическое «или»: хотя бы одно выполнено

```
        (5 < 3) || (7 == 7) — истина
```

`!` — отрицание: `!(2 * 2 == 5)` — истина

Арифметические операторы

`+ - * /` — аргументы могут быть любыми; результат целый $\Longleftrightarrow$ оба аргумента целые

`%` — остаток от деления (сохраняет знак числа)

Оператор присваивания

`=` (например, `a = b + c;`)

Операторы сравнения

`< <= == > >= !=` (не путайте `=` и `==`)

Результат сравнения — 0 («ложь») или 1.

Условная инструкция (ветвление)

```
if (условие)
        инструкция;
else /* необязательно */
        инструкция;
```

Несколько инструкций можно объединить в одну при помощи *операторных скобок* `{ }`.

Инструкция выбора

```
switch (выражение)
{
   case значение: инструкции; ... break;
   ...
   default: инструкции;
}
```

Повторяющиеся вычисления (циклы)

Цикл с предусловием:

```
while (условие)
        инструкция;
```

Цикл с постусловием:

```
do
        инструкция;
while (условие);
```

Самый общий вариант цикла:

```
for (инициализация; условие; модификация)
        инструкция;
```

Пример:

```
for (i = 0; i < N; i++)
        printf("%d\n", i);
```

Полезные операторы

`i++;` — увеличение переменной `i` на единицу

`i--;` — уменьшение переменной `i` на единицу

`i += k;` — увеличение переменной `i` на `k`

`i -= k;` — уменьшение переменной `i` на `k`

(аналогично для других операций: `*=`, `/=`, `%=`)

Полезные функции из math.h

`sqrt(x)` — квадратный корень

`fabs(x)` — модуль вещественного числа

`sin(x)`, `cos(x)` — синус, косинус

При использовании функций из `math.h` необходимо при компиляции указывать ключ *-lm*.

Редактирование, компиляция и запуск программы в Linux

`vim` *file.c* — запуск текстового редактора

Insert — режим ввода, Esc Shift+Z Z — сохранение и выход

`gcc -o` *file file.c* — компиляция программы *file.c* в исполняемый файл *file*.

Если нужно линковать программу с дополнительными библиотеками (например, с математической при использовании `math.h`), нужно добавить ключ *-lm*:

`gcc` *-lm* `-o` *file file.c* — компиляция с использованием математической библиотеки.

./file — запуск программы *file* из текущего каталога.

Программа на языке C состоит из функций, каждая из которых имеет некоторый (возможно, пустой) упорядоченный набор аргументов и, возможно, *возвращает* в качестве результата значение некоторого типа. Исполнение программы начинается с функции `main`, результат которой имеет тип `int`; обычно при корректном завершении программы результат функции `main` должен быть равен нулю.

Напишем функцию, вычисляющую модуль целого числа. Её результатом будет также целое число.

```
int abs(int x)
{
    if (x < 0)
        return (-x);
    else
        return x;
}
```

Т. к. `return` завершает выполнение функции, ключевое слово `else` («иначе») можно опустить: в случае, если число `x` отрицательно, инструкция `return (-x)` завершит выполнение функции; в противном случае (даже без `else`) будет выполнено `return x`.

```
int abs(int x)
{
    if (x < 0)
        return (-x);
    return x;
}
```

Заметьте, что функция `abs` ничего не печатает на экран. Намного важнее то, что она возвращает значение при помощи `return`. Напечатанное видно только человеку, в то время как результат функции можно использовать в программе:

```
int main()
{
    int x = -5;
    int y = abs(x); /* присвоили результат функции переменной */
    printf("%d\n", y);
    printf("%d\n", abs(11)); /* использовали результат как аргумент для другой функции */
    return 0;
}
```

Для печати информации на экран (точнее, в поток стандартного вывода, им может оказаться и файл) используется функция `printf`. Её первый аргумент — строка в двойных кавычках — печатается, при этом

вместо управляющих последовательностей `%d`, `%f` и прочих подставляются остальные аргументы (подряд). Функция `printf` сама не может определить, какого типа аргументы ей переданы, потому и приходится указывать тип при помощи различных (для каждого типа своя) управляющих последовательностей.

Мы пока не будем использовать функцию `scanf`, которая вводит значения переменных с клавиатуры (более строго — из потока стандартного ввода), до её использования необходимо ознакомиться с понятием указателя.

Как вы уже могли заметить, переменные в программе могут быть объявлены в начале функции (например, в предыдущем примере в функции `main` объявлены две переменных `x` и `y`). Кроме того, их можно объявлять в начале любого блока `{ }`. Зачастую компиляторы разрешают вставлять определение переменной в любое место программы, как требует более новый стандарт языка C.

3 Задачи для решения у доски

Задача. Реализуйте функцию `max`, возвращающую наибольшее из двух чисел.

Задача. Реализуйте функцию `max3`, возвращающую наибольшее из трёх чисел.

Задача. Решите квадратное уравнение $ax^2+bx+c=0$, коэффициенты которого заданы в функции `main`:

```
double a = 2.0, b = 6.0, c = -8.0;
```

Задача. Напишите функцию, проверяющую, является ли данный год високосным. Год является високосным, если он или делится на 400, или делится на 4, но не на 100. Например, 2000 и 2004 года — високосные, а 1900 – нет.

Задача. Вычислите сумму всех натуральных чисел от 1 до N при помощи цикла `for` и при помощи цикла `while` (забудем на время, что существует формула суммы арифметической прогрессии).

Задача. Подсчитайте количество цифр в натуральном числе N.

Задача. Проверьте, является ли данное число простым. Реализуйте проверку в отдельной функции `is_prime`, возвращающей 0 для составного и 1 для простого числа. Эта функция не должна ничего печатать на экран.

Задача. Используя функцию `is_prime`, напечатайте все простые числа от 2 до 1000.

4 Рекурсивные функции

Рекурсивной называется функция, вызывающая сама себя напрямую (в её определении используется вызов её самой) или косвенно (например, первая функция вызывает вторую, которая, в свою очередь, вызывает первую). Рекурсия является очень мощным инструментом в программировании, которым, впрочем, необходимо уметь пользоваться.

Простейшим примером рекурсии является *хвостовая рекурсия*, в которой рекурсивный вызов появляется в последней инструкции функции. Классическим примером рекурсивной функции с хвостовой рекурсией является функция вычисления факториала[1] натурального числа n:

Рекурсивный вариант

```
int fact(int n)
{
    if (n == 0)
        return 1;
    return (n * fact(n - 1));
}
```

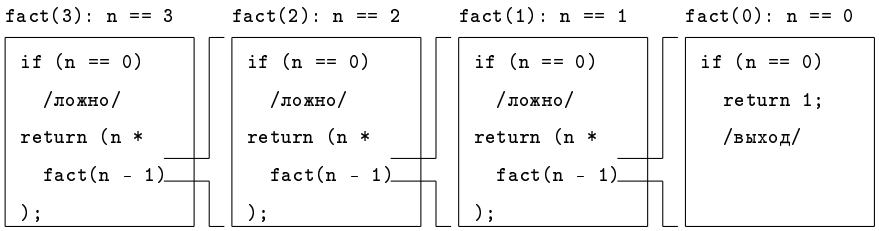
Итеративный вариант

```
int fact(int n)
{
    int result = 1;
    while (n > 1)
    {
        result *= n;
        n--;
    }
    return result;
}
```

Рассмотрим подробнее, как работает рекурсивная реализация в случае вызова `fact(3)`. Даже если вы полностью поняли предыдущий при-

[1]На самом деле приведённый здесь пример хвостовой рекурсией в строгом смысле **не является**, так как «настоящее» определение хвостовой рекурсии несколько сложнее. Эту тему мы подробнее обсудим на следующих семинарах.

мер кода, пожалуйста, рассмотрите рисунок.



Важно понять два основных момента:

1. На момент вычисления каждого следующего вызова `fact` в памяти находятся все предыдущие (т. е. сохраняется значение переменных из предыдущих функций) — следовательно, рекурсия использует заведомо больше памяти, чем итеративный способ.

2. При завершении любой функции (в том числе рекурсивной) вычисление продолжается с той точки, в которой был совершён вызов функции.

Хвостовой рекурсии всегда можно сопоставить аналогичную функцию, реализованную без рекурсии и использующую ограниченное (не зависящее от глубины рекурсии) количество переменных. Из-за того, что хвостовая рекурсия неэффективна по памяти, её следует избегать. Однако, существуют более сложные формы рекурсии, которые нельзя преобразовать в итеративную программу, использующую заранее фиксированное количество переменных. На следующих занятиях мы рассмотрим примеры таких функций.

5 Проверочное задание («пятиминутка»)

Тема: функции, простые циклы. Формулировки заданий по вариантам:

1. Реализуйте функцию, которая принимает параметром натуральное число N и возвращает число 2^N.

2. Реализуйте функцию, которая принимает параметром натуральное число N и возвращает сумму квадратов всех чисел от 1 до N.

3. Реализуйте функцию, которая принимает параметром натуральное число N и возвращает число 3^N.

4. Реализуйте функцию, которая принимает параметром натуральное число N и возвращает сумму кубов всех чисел от 1 до N.