Lecture_01-Introduction

> **В лекции**
>
> 1. Введение в программирование на примере языка C.

Введение в программирование на примере языка C

Перед тем как начинать изучать тонкости современных подходов к написанию сложных и больших программ, требуется определиться с инструментом, который позволит решить поставленную задачу. Более того, мы накладываем условия на сам этап разработки, просто решить поставленную задачу мало, мы хотим решить задачу за минимально приемлемое время и чтобы решение было как можно более эффективно. Тут возникает первый вопрос: "Почему бы не использовать Python? Зачем вообще программировать на C/C++,C#, если все требуемые для большинства проблем библиотеки давно уже написаны и с легкостью устанавливаются в Python?". Начнем с того, что библиотеки разрабатываются под конкретные задачи и обхватить все случаи не способны, значит что-то разрабатывать так или иначе придется. Во-вторых, производительность интерпретируемых языков меньше (коем и является язык python). В-третьих, программирование станет гораздо проще, когда вы поймете как все устроено на фундаментальном уровне.

> Интерпретируемые языки компилируются в момент выполнения программы, в то время как компилируемые языки делаю это заранее.

> Хороший пример понимания основ программирования -- приращение аргумента приращения в цикле `for` через инкремент, а не пост-инкремент.

Программирование в первую очередь является инструментом создания, выучить язык программирования как правило недостаточно, а в некоторых случаях выучить его и вовсе невозможно. Чтобы программировать, нужно освоить практики, которые способствуют удобной разработке и научиться читать/понимать/писать код, который лежит в основе вашей любимой Python-библиотеки. Только так можно действительно изучить технологию, научиться воспроизводить ее и создавать на ее основе что-то новое.

Язык C.

Язык C -- это небольшой, но очень выверенный и продуманный язык программирования, который был разработан в 1972 году для работы над операционной системы UNIX (от которой пошло целое семейство unix-подобных операционных системы, самые известные: MacOS, Linux, FreeBSD). Тем не менее, сегодня C не имеет привязки к определенной операционной системе или аппаратной платформе (ядро для операционной системы Windows -- WindowsNT написано на языке C). Все это обусловлено тем, что язык C обладает высокой производительностью и контролем над процессом разработки. Программист напрямую управляет оперативной памятью, а сам код напрямую компилируется в машинный код (чего не могут себе позволить интерпретируемые языки или языки с Jit-компиляцией в байт-код).

> Дополнительно язык C позволяет писать модульные программы, что и будет предметом наших занятий (почти).

Hello World и минимум о функциях

> Нет лучшего подхода начинать изучать программирование, чем начать писать программы.

```c
#include <stdio.h> // Включение заголовочного файла

main() // Определение функции main.
{
```

```
    // Тело функции
    // ...

    //printf(формат, аргументы);
    printf("hello, world\n");
}
```

> Такая программа не очень интересна, но в ней все-равно есть что рассмотреть.

```
#include <stdio.h>
```

В данной строке происходит подключение **заголовочного файла стандартной библиотеки C** (Что по сути не является самим языком C из коробки, а является его конкретным, в зависимости от системы, расширением).

> Для нашего предмета заголовочные файлы важны тем, именно они, в некотором роде, определяют доступ к разработанным нами модулям.

В этой конструкции фигурируют два важных понятий: **интерфейс** и **реализация интерфейса**. Заголовочный файл `stdio.h` не содержит кода (на данном этапе изучения для нас по другому быть не может), который мог бы вывести строковую константу `"Hello, world!\n"` в поток вывода (вывод на экран в интерпретаторе команд или файл), но содержит определение функции `printf(const char* format, ...)`. На этапе определений никакой реализации еще не существует. Компилятор, который мы используем для создания исполняемого файла, и который специально разработан для нашей системы, знает о существовании специальной библиотеки, которая реализует заголовочный файл `stdio.h` и которая где-то лежит в нашей системе.

> На первый взгляд довольно сложный и не понятный механизм работы. Но он позволяет писать программы, код которых будет одинаков как для Linux, так и для Windows. Все потому, что ввод/вывод символов куда-либо в операционной системе зависит от того, как эта операционная система сама по себе работает (механизм вывода символов в MacOS и Windows различны). Но эта "игра" с подстановкой реализаций позволяет писать абсолютно переносимый код, а главное дебри вывода символов конкретной операционной системы скрыты от нас.

Минимум о функциях

Программа на C независимо от ее размера состоит из функций и переменных. Функция содержит **операторы** -- команды для выполнения определенных вычислительных операций, а в переменных хранятся числа и другие данные, используемые в этих операция.

Для определения функции необходимо указать ее **название** (название должно отражать логику функции), **передаваемые аргументы в скобках**, **границы самой функции**. В C-подобных языках граница определяется фигурными скобками.

- Название
- Аргументы
- Границы

Объявление функции задает сигнатуру(типы аргументов и тип значения)

```
int foo(int x, double y);
void bar(); // Функция не возвращает значения
```

Функция может быть вызвана даже если еще не определена

```
// Где-то в программе (но только один раз) должно найтись тело функции.
int t = 42; int s; s = foo(t, 1.0);
```

```
// Возвращаемое значение можно проигнорировать даже если не не void.
int t = 42; foo(t, 1.0); //ok
```

Базовые стандартные функции

Мы можем определить сколь угодно функций, но в C-подобных языках программирования функция `main(...)` является особым случаем. Любая программа начинается с функции `main(...)`.

Функция main() это точка входа вашей программы, с неё начинается исполнение кода.

- Она должна быть ровно одна.
- По конвенции при правильном исполнении программы main возвращает ноль.
- Функция printf (вывод) и scanf (ввод) импортируются из `libc` и их объявления находятся в заголовочном файле stdio.h

Суть использования функций заключается в умышленном сокрытии кода (инкапсуляция) за некоторым псевдонимом (названием функции). Если мы знаем, что по выполнению функции случиться, то нам необязательно помнить то, как она работает. Также выделение некоторого кода в функцию облегчает процесс восприятия смысла и структуры кода.

Минимум об указателях

Указатель на переменную это способ косвенно записать или прочитать ее значение.

```
int a = 0;
int *pa = &a // получаем адрес переменной в памяти

a = 1; // прямая запись, теперь a == 1 и *pa == 1
*pa = 2; // косвенная запись, теперь a == 2 и *pa == 2
```

Теперь попробуем написать что-то более сложное

```
#include <stdio.h>
#include <stdlib.h> // Для функции оборт.

void read_input(int *, int *); // Определение функции проверки ввода

main() {
  int a, b, p, q;
  read_input(&a, &b); // Вызов функция проверки ввоада.
  p = a / b;
  q = a % b;
  printf("p: %d, q: %d\n", p, q);
}

// Тело функции проверки ввода
void read_input(int *pa, int *pb) {
  int nitems;
  printf("input a and b: ");
  nitems = scanf("%d %d", pa, pb);
  if (nitems != 2 || *pb == 0) { // проверка ввода, b не должно быть равно 0
    printf("Error: input invalid, expect any a and b != 0\n");
    abort();
  }
}
```

В данной программе мы пытаемся решить задачу нахождения частного и остатка. Для всех a и b, найдутся такие p и q, что $a = b * p + q$.

```
> gcc divmod.c -o divmod
> ./divmod
input a and b: 10 3
p = 3, q = 1
```

> Еще раз обратите внимание на способ создания и вызова исполняемого файла через компилятор gcc.

Новая функция `abort()` нужна чтобы прервать программу в произвольной точке, её объявление находится в stdlib.h.

Минимум о циклах

В языке C есть три основных типа циклов: `for`, `while`, `do-while`.

- Циклы `while:` выполняются пока какое-то условие истинно.

```
while(i < 100) { /* тело цикла */ }
```

- Циклы `do-while`: точно выполняется единожды, т.к. тело цикла находится до условия

```
do ( /* тело цикла */ ) while (i < 100)
```

- Циклы `for`: содержат инициализацию, условие, приращение переменной

```
for (int i = 0; i < 100; ++i ) { /* тело цикла */ }
int i = 0; while( i < 100) { /* тело цикла */; ++i; }
```

Программа вычисления температура по Фаренгейту и Цельсию.

```
#include <stdio.h>

main()
{
    int fahr, celsius;
    int lower = 0, upper = 300, step = 20;

    fahr = lower;
    while(fahr <= upper) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + step;
    }
}
```

> Данная программа выводит таблицу температуры по Фаренгейту и их соответствий по шкале Цельсия.

Рассмотрим программу:

- Комментарии содержат краткое описание того, что делает программа. Любые символы после `//` или `/* */`, игнорируется компилятором.
- Переменные -- некоторые структуры в коде для хранения данных. Обычно переменные объявляется до любых выполняемых операторов (обычно это в начале функции). Также в строго-типизированных языках программирования требуется указывать тип данных перед объявлением переменной. Самые часто используемые типы данных: `int` -- целые числа, `float` -- вещественные числа с плавающей точкой, `double` -- вещественное число с двойной точностью, `char` -- символ. Ключевым моментом в использовании типов данных является их размер, если необходимо записать очень большое число, то оно

точно не поместить в тип `int`, диапазон которого от -2 147 483 648 до 2 147 483 647 (при 32-битном разряде), поэтому можно использовать тип большего размера, например, `long`.

- Арифметические выражения представляют собой базовые математические операции. Они позволяют получать новые данные на основе уже полученных. Символы, представляющие математические операции (**сложение**, **вычитание** и т.д.) **называются операторами**.
- Циклы -- управляющая конструкция, которая позволяет выполнять один и тот же фрагмент кода многократно до тех пор, пока не выполниться некоторое условие. Циклы также содержат в себе операторы, в данном примере содержится оператор сравнения **меньше-или-равное**: `<=`.
- Форматированный вывод -- способ задания стиля вывода символов в поток вывода. Для того, чтобы компилятор мог различать символы на вывод и символы, отвечающие за форматирование, вводятся **управляющие последовательности (escape sequence)**:
  - `\n` -- перенос строки;
  - `\t` -- табуляция или четыре пробела;
  - `\b` -- возврат на один символ назад с затиранием;
  - `\"` -- двойная кавычка;
  - `\\` -- обратная косая черта.

Однако любую программу можно переписать короче, в данном случае мы можем упростить код используя цикл for.

Цикл `for` во много является обобщением цикла `while`. Цикл `for` состоит, как правило, из трех выражение: **инициализация**, **условия**, **приращения шага**. Выбор между `while` и `for` определяется соображениями ясности программы. Цикл `for` более удобен в тех случаях, когда инициализация и приращение шага логически связаны друг с другом общей переменной и выражаются единичными инструкциями, что позволяет значительно сократить код.

```
#include <stdio.h>

main()
{
    int fahr;
    for (fahr = 0, fahr <= 300; fahr = fahr + 20) {
        printf("%d %d\n", fahr, (5.0/9.0)*(fahr-32));
    }
}
```

Цикл `for(...)` может быть записан как:

1. `for(int a = 0; a < n; ++a) {};`
2. `for (;a < n;) {}` -- как while;
3. `for (;;++a) {};`
4. `for (int a = 0;;) {};`
5. `for(;;)` -- бесконечный цикл.

Условные конструкции

Условные конструкции позволяют проверять выражения на истинность или ложность и выполнять соответствующие результату проверки операторы:

```
if (выражение) {
    оператор1;
}
else
    оператор2;
```

> Выполняются всегда только один оператор из двух, ассоциированных с конструкцией `if-else`. Если выражение `Правда`, выполняется `оператор1`, в противном случае выполняется `оператор2`.

Также существует конструкция `if-else if-else`, которая позволяет накладывать дополнительное-альтернативное условие на выражение, если проверка в конструкции `if(...)` определяет выражение как Ложное.

```
if (выражение1) {
    оператор1;
}
else if (выражение2) {
    оператор2;
}
else
    оператор3;
```

> В остальном работа аналогична конструкции `if-else`.

Минимум о типах данных

Фундаментальные объекты данных, с которыми работает программа, -- это переменные и константы. Для используемых в программе переменных необходимо указать их **тип**, а также иногда их **начальные значения**. Манипуляции и преобразования над данными выполняются с помощью знаков операций. Переменные и константы объединяются в выражения, чтобы таким образом порождать новые значения.

В языке С существуют все несколько базовых типов данных:

- char -- один байт, содержащий один символ из локального символьного набора;
- int -- целое число, обычно имеющее типовой размер для целых целых чисел в данной системе;
- float -- вещественное число одинарной точности с плавающей точкой;
- double -- вещественное число двойной точности с плавающей точкой.

Для работы с разными типами, функции printf и scanf используют **форматные спецификаторы.**

Тип	Форматные спецификатор
char	"%c"
int	"%d"
float	"%f"
double	"%d"

Домашнее задание

Дана программа простого калькулятора. Необходимо ее переписать так, чтобы были доступны оператор `/` и `*`. Также необходимо добавить соответствующие проверки на ввод. Программа не должна завершаться ошибкой при делении на 0, необходимо проверять одно из значений. При защите лабораторной работы необходимо объяснить код преподавателю. На защите может быть спрошена любая строка кода.

```c
#include <stdio.h>
#include <stdlib.h>

void read_input(int *, char *, int *);

int main()
{
    char op_;
    int a, b, n;
```

```
    read_input(&a, &op_, &b);

    if (op_ == '+') {
    n = a + b;
    }
    else if (op_ == '-') {
        n = a - b;
    }
    printf("result: %d\n",n);
    return 0;
}

void read_input(int *pa, char *pop_, int *pb)
{
    printf("Input a, operator (+ or -), b: ");
    int nitems = scanf("%d %c %d", pa, pop_, pb);
    if (nitems != 3) {
        printf("Error: input invalid.\n");
        abort();
    }
    if (*pop_ != '+' && *pop_ != '-') {
        printf("Error: input invalid, operator input error.\n");
    abort();
    }
}
```

> Ссылка на репозиторий: https://github.com/llfedotov73/smts-smd/blob/main/labworks/calc-base.c

Студенту необходимо создать учебный репозиторий на GitGub и размещать в нем исходный код своих лабораторных.