Конспект лекций курса ИУ-7
“Программирование на языке С”

Жихарев Кирилл ИУ7-24Б

Содержание

1 Особенности языка Си

Основные концепции языка Си

- это язык сравнительно «низкого» уровня;
- это «маленький» язык с однопроходным компилятором;
- предполагает, что программист знает, что делает.

Как заметил пользователь Habr-а *lesha_penguin*:

> ... язык C – это язык, практически не мешающий программисту делать то, что он решил делать. В нем нет “няньки”, которая ходит за тобой, и пытается тебе “мешать делать глупости”. Все равно от всех глупостей не застрахуешь.
> ...
> Язык C – наверное единственный язык из высокоуровневых, который вам дает практически неограниченную свободу. Причем плата за эту практически полную свободу смешная – всего-навсего ответственность!

1.1 Отличия языка Си от Python

- Python – интерпретируемый язык, C – компилируемый язык.
- Python – язык с динамической типизацией, C – язык со статической типизацией.
- Python – автоматическое управление памятью, C – ручное управление памятью.

1.2 Пример программы на языке Си

Программа “Hello, world!” на языке Си:

```
/*
 * Программа на языке Си
 */
#inclide<stdio.h>

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

1.3 Выполнение программы из примера

Для начала программу требуется *скомпилировать*:

```
gcc -std=c99 -Wall -Werror -o hello.exe hello.c
```

Здесь:

- `gcc` – название компилятора;
- `-std=c99` – используемый стандарт;
- `-Wall` – включение всех предупреждений;
- `-Werror` – трактовать предупреждения как ошибки;
- `-o hello.exe` – имя исполняемого файла;
- `hello.c` – что компилировать.

1.4 Выводы

- Программа на языке Си состоит из одной и более функций.
- Среди этих функций обязательно должна быть функция с именем main, с которой начнётся выполнение программы.
- Значение, которое возвращает функция main, говорит об успешности работы программы.
- Если в программе используется какая-либо функция из стандартной библиотеки, компилятору необходимо сообщить ее описание, подключив соответствующий заголовочный файл.
- Игнорировать предупреждения компилятора нельзя.

2 Переменная в языке Си

Определение 2.1. Переменной называется именованный участок памяти, обладающий некоторым типом.

В Си *переменная* перед использованием должна быть *определена*.

```
int x;     // Определение
x = 42;
```

Замечание. Переменная в языке Python сильно отличаются от переменных в языке Си!

2.1 Выбор типа переменной

Выбор правильного типа для переменной очень важен, потому что тип определяет

- как переменная хранится;
- какие значения может принимать;
- какие операции могут быть выполнены над переменной.

Например, целочисленный тип `int` может принимать ограниченное количество значений (от $-2^{15}+1$ до $+2^{15}-1$). Тип числа с плавающей точкой (ЧПТ) может принимать значительно больший диапазон значний, а также дробные числа. Однако операции с такими числами выполняются дольше.

2.1.1 Пример использования переменных

```c
#include <stdio.h>

int main(void)
{
  int a, b, c;

  printf("Enter two integers:\n");
  scanf("%d %d", &a, &b);

  c = a + b;

  printf("a + b = %d\n", c);

  return 0;
}
```

2.2 Выводы

- Программы обрабатывают данные, которые представлены переменными.
- Язык Си это язык со статической типизацией. Прежде чем использовать переменную, ее нужно описать.
- Описание переменной состоит из указания типа переменной и ее имени.
- Выбор правильного типа для переменной очень важен.
- Для вывода значений переменных обычно используется функция printf, работа которой управляется строкой форматирования.
- Для ввода значения переменных обычно используется функция scanf, работа которой управляется строкой форматирования.
- Алгоритм работы функции scanf основана на «сопоставлении с образцом».
- Функция scanf должна изменять значения переменных, поэтому в качестве параметров получает адреса этих переменных.

3 Функции

3.1 Подпрограмма

Определение 3.1. *Подпрограмма* – именованная часть программы, содержащая описание определённого набора действий. Подпрограмма может быть многократно вызвана из разных частей программы.

Преимущества использования подпрограмм:

- Уменьшение сложности программирования за счёт декомпозиции задачи.
- Уменьшение дублирования код.

- Возможность повторного использования кода.
- Сокрытие деталей реализации от пользователей подпрограммы.

3.1.1 Виды подпрограмм

Определение 3.2. *Функция* – это подпрограмма специального вида, которая всегда должна возвращать результат. Вызов функции является выражением.

Определение 3.3. *Процедура* – это независимая именованная часть программы, которую после однократного описания можно многократно вызвать по имени из последующих частей программы для выполнения определенных действий.

3.1.2 Структура подпрограммы

Любая подпрограмма состоит из двух частей: *заголовка* и *тела подпрограммы*. Заголовок описывает информацию, необходимую для вызова подпрограммы. Тело подпрограммы – набор операторов.

3.1.3 Вызов подпрограммы

Вызов подпрограммы выполняется с помощью команды вызова, включающей в себя имя подпрограммы, за которым следуют параметры.

Чтобы отличать параметры подпрограммы, описанные в её заголовке и теле, от параметров, указываемых при вызове подпрограммы, используются *формальные* и *фактические* параметры. Формальные параметры указываются при описании подпрограммы, а фактические – непосредственно при её вызове. Фактические параметры часто называют *аргументами*.

3.2 Передача параметров

Параметры могут передаваться *по значению* и *по ссылке*.

При передаче параметра по значению значение аргумента используется для инициализации формального параметра. Таким образом, изменение формального параметра внутри подпрограммы *не влияет на вызывающий код*.

При передаче параметра по ссылке, подпрограмма получает доступ к ячейке памяти, в которой находится аргумент. Изменение формального параметра в таком случае повлияет на вызывающий код.

3.3 Функции в языке Си

3.3.1 Возвращаемое значение

- Функция может вернуть значение любого типа кроме массива.
- Если функция ничего не возвращает, то в качестве возвращаемого значения следует указать `void`.
- Для возврата значения используется оператор `return`.

3.3.2 Параметры функции

- Любая функция может принимать параметры.
- Перед именем каждого параметра указывается его тип. Тип указывается перед каждым параметром, даже если несколько параметров имеют один и тот же тип!
- Описания параметров в списке разделяются запятыми.
- Если у функции нет параметров, вместо списка параметров указывается ключевое слово `void`.

3.3.3 Тело функции

- У каждой функции есть исполнимая часть, которая называется *телом функции* и заключена в фигурные скобки.
- Тело функции может содержать как описания переменных, так и операторы. Переменные, описанные в теле функции, "принадлежат" только этой функции и не могут быть ни получены, ни изменены другой функцией.
- Тело функции не может содержать в себе определения других функций.

Оператор *return* прерывает выполнение функции и возвращает вычисленное значение и управление в ту часть программы, из которой эта функция была вызвана. Его можно использовать в функциях, которые ничего не возвращают.

3.3.4 Объявление функции

Объявление функции предоставляет компилятору всю информацию, необходимую для вызова функции: количество и типы параметров, их последовательность, тип возвращаемого значения.

Объявление функции состоит из заголовка функции

Пример объявления функции:

```
double average(double a, double b)
```

3.3.5 Объявление и определение

Объявление (*declaration*) – это инструкция компилятору, как использовать указанное имя (описывает свойства переменной или функции). Объявлений одного и того же имени может быть сколько угодно, главное чтобы они все были согласованы (т.е. одинаковы).

Определение (*definition*) осуществляет привязку имени к сущности (к памяти для данных или к коду для функций), т.е. специфицирует код или данные, которые стоят за этим именем. В языке Си существует правило единственного определения. Это означает, что определение может быть только одно.

Пример:

```
// Объявление
double average(double a, double b);

...

// Объявление
double average(double a, double b)
{   // Определение
  return (a + b) / 2
}
```

3.3.6 Функции без параметров

`void foo(void)` – функция, которая точно не принимает ни одного параметра.
`void foo()` – функция может принимать любой (в том числе и нулевое) количество параметров.

3.3.7 Чистые функции

Определение 3.4. *Чистой функцией* называется функция, которая удовлетворяет следующим условиями:

- является детерминированной;
- не обладает побочными эффектами.

Определение 3.5. Функция называется `детерминированной`, если для одного и того же набора входных данных она возвращает один и тот же результат.

Определение 3.6. Функции с побочными эффектами – это функции, которые в процессе выполнения своих вычислений могут модифицировать значения глобальных переменных, осуществлять операции ввода/вывода.

3.3.8 Вызов функции

1. Выделяется память.
2. Создаются локальные переменные и параметры.
3. Параметрам присваиваются переданные в функцию значения.
4. Инициализация локальных переменных.
5. Выполнение тела функции.
6. Значние возвращается из функции.
7. Освобождается память.

3.3.9 Выводы

- Параметры-переменные и локальные переменные отличаются только одним: параметрам-переменным автоматически присваиваются начальные значения, равные значениям в точке вызова функции.
- В функции изменяется параметр-переменная, а не переменная, переданная в качестве аргумента. Функция "забывает" о переменной-аргументе сразу же после инициализации параметра-переменной.
- Имена параметров-переменных могут совпадать с именами переменных, передаваемых в качестве аргументов.

3.4 Рекурсия

Определение 3.7. *Рекурсией* называется явление, когда функция вызывает саму себя.

Вычисление факториала как пример рекурсии:

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

3.4.1 Преимущества и недостатки использования рекурсии

Преимущества

- Рекурсивная форма может быть структурно проще и нагляднее, в особенности, когда сам реализуемый алгоритм рекурсивен.

Недостатки

- Рекурсивный вызов использует больше памяти, поскольку создает свой набор переменных.
- Рекурсия выполняется медленней, поскольку на каждый вызов функции требуется определенное время.

3.4.2 Хвостовая рекурсия

Определение 3.8. *Хвостовой рекурсией* называется частный случай рекурсии, при котором любой рекурсивный вызов является последней операцией перед возвратом из функции.

Подобный вид рекурсии может быть легко заменён на итерацию путём формальной и гарантированно корректной перестройки кода функции. Оптимизация хвостовой рекурсии путём преобразования её в итерацию реализована во многих оптимизирующих компиляторах.

4 Одномерные статические массивы

Определение 4.1. *Массивом* называется последовательность элементов одного типа, расположенных в памяти последовательно.

4.1 Массивы в Си

Статический массив объявляется как:

```
int arr[10];
```

- Тип массива может быть любым.
- Количество элементов является целочисленным константным выражением.
- Размер массивы не может быть изменён в ходе выполнения программы.

4.1.1 Доступ к элементу массива

Доступ к элементу массива осуществляется операцией *индексации*.

```
int arr[10];
...
int a = arr[2]; // Третий элемент массива
```

- Для доступа к элементу массива используется индекс.
- Индексация выполняется с нуля.
- В качестве индекса может выступать целочисленное выражение.
- Си не предусматривает никаких проверок на выход за пределы массива.

4.1.2 Инициализация массива

```
arr[] = { 1, 2, 3 };     // { 1, 2, 3 }
arr[5] = { 1, 2, 3 };    // { 1, 2, 3, 4, 5 }
arr[5] = { 0 };          // { 0, 0, 0, 0, 0 }
arr[5] = { [1] = 1, [3] = 3 }; // {0, 1, 0, 3, 0 }
```

5 Указатели

Определение 5.1. *Указатель* – это объект, содержащий адрес объекта или функции, либо выражение, обозначающее адрес объекта или функции.

5.1 Указатели в языке Си

Объявляется как обычная переменная, но перед именем указывается знак “астерикс” (`*`). Пример: `int *p;`

5.1.1 Разновидности указателей

- Типизированный указатель.
- Бестиповый указатель (`void *p`).
- Указатель на функцию.

5.1.2 Зачем нужны указатели?

- Передача изменяемых параметров в функцию.
- Передача объёмных параметров; считаем параметр объёмным, если он занимает больше памяти, чем `double`.
- Динамическое выделение памяти.
- Реализация ссылочных типов данных.

5.1.3 Операции с указателями

Операция получения адреса – возвращает адрес переменной в памяти. Обозначается как знак `&` перед именем переменной. Пример:

```
int a = 5;
int *p = &a;
```

Операция разыменования – возвращает значение переменной по адрес. Обозначается как знак `*` перед именем переменной. Пример:

```
int a = 5;
int *p = &a;
int b = *p; // 5
```

> **Замечание.** Типичная ошибка: знак `*` принадлежит ближайшему имени переменной. То есть:
>
> ```
> int *a, b;  // a - указатель, b - целочисленная переменная
> int a = 5, *p = &a; // можно сделать так
> ```

Размер указателя одинаковый в рамках машины.

Операции `*` и `&` – взаимно обратные. То есть выражение `&*a` превращается в просто `a`.

5.1.4 Инициализация указателей

Указатель может инициализироваться:

- адресом переменной того же типа;
- значением указателя того же типа;
- значением `NULL`;

5.1.5 Указатели и `const`

Следует различать два способа использования ключевого слова `const` с указателями.

1. Указатель на константу: `const int *p`. Нельзя изменить значение, на которое указывает указатель.
2. Константный указатель: `int *const p`. Нельзя изменить адрес переменной, на которую ссылается указатель.

Можно комбинировать два вида использования `const` и получить *константный указатель на константу*, который совмещает свойства двух предыдущих.

5.2 Указатели и массивы

Результат выражения, состоящего из имени массива, представляет собой адрес области памяти, выделенной под этот массив (англ. *"array decay to pointer"*)

```
int a[10], *pa;
pa = a;
pa = &a[0];
```

5.2.1 Исключения

1. `sizeof` для массива возвращает его размер, а для указателя – размер переменой, содержащей адрес.
2. При взятии адрес от массива, вернётся значение типа *указатель на масссив*, который будет численно равен адресу массива.
3. Строковый литерал-инициализатор массива `char[]`.

5.2.2 Различия указателя и массива

- Массиву нельзя присвоить другой адрес.
- Под переменные выделяется разное количество памяти.

5.2.3 Передача массива в функцию

Любое объявление параметра, похожее на массив, трактуется компилятором как указатель. Это значит, что следующий код абсолютно валиден:

```
void foo(int a[10])
{ ... }

int arr[25] = { ... };
int a = 5, *pa = &a;
foo(arr);
foo(pa);
```

Из этого следует, что функция *не может узнать размер массива через `sizeof`*.

5.2.4 Тип `size_t`

`size_t` – беззнаковый целый тип, предназначенный для представления размера любого объекта в памяти. Размер типа определяется относительно возможностей аппаратной платформы. Тип определен в заголовочном файле `stddef.h`.

Максимальный размер `size_t` указан в константе `SIZE_MAX`, определённой в `stdint.h`

5.2.5 Сложение указателя с числом

Запись вида:

```
int a, *pa = &a;
int pb = pa + n;
```

. Обозначает операцию: *адрес* `pb` *равен адресу* `pa` *плюс n размеров переменной-типа указателя.*

То есть можно записать как:

```
int arr[10] = { ... };
arr + 2;      // arr[2]
```

5.2.6 Операция индексации

Компилятор заменяет оператор индексации `arr[i]` на выражение вида `*(arr + i)`

5.2.7 Сравнение указателей

Указатели можно сравнивать:

- одного типа;
- с `NULL`.

5.2.8 Разность указателей

Указатели можно вычитать. Разница будет представлена типом `ptrdiff_t`. Данный тип определён в заголовочном файле `stddef.h`

```
int arr[];
ptrdiff_t pd = arr[0] - arr[2];  // -2
```