Основы программирования

Лекция № 3, 16 марта 2017 г.

http://xkcd.ru/371

Указатели (продолжение)

Указатель, никуда не указывающий

`NULL` — специальное константное значение, символизирующее, что указатель не указывает ни на какую память. Объявлено в заголовочном файле `stdlib.h`.

```
int* ptr = NULL;
int value = *ptr; // run time error
*ptr = 37; // run time error
```

Разные названия, но суть одна (segmentation fault, segfault, access violation, «Программа выполнила недопустимую операцию...», ...).

Численное значение NULL == 0.

Указатель, указывающий хоть куда

`void*` — специальный тип указателя, который может указывать на любые данные в памяти. Может быть приведен к любому другому типу указателей и обратно.

```
double x = 37;
double* px = &x;
void* p = px;
int* py = p;
```

Минутка философии: «Какова природа void?» — спросил учитель, ...
`http://thecodelesscode.com/case/5?lang=ru`

Указатель трактует указываемое содержимое как int, хотя на самом деле там находится double — это потенциальная ошибка.

Обязательно прочитать рассказ по ссылке.

Динамическая память

Преимущества динамической памяти

- Выделяется и освобождается динамически по запросу программы.
- Размер задается динамически.

Внутри функции нельзя создать массив, который будет существовать после выхода из нее.

Размер массива должен быть известен на момент компиляции программы.

Самый гибкий и самый сложный в работе вид памяти.

Есть еще автоматическая и статическая память, но их описание выходит за рамки данной лекции. Их основные недостатки: фиксированный размер и недостаточно гибкое время жизни.

Выделение блока памяти

```
void* malloc(size_t size);
```

Функция выделяет блок памяти размером `size` байт и возвращает указатель на начало блока. В случае, если память выделить не получилось, возвращает NULL. Объявлена в заголовочном файле `stdlib.h`.

malloc = Memory ALLOCate

size_t - беззнаковый целочисленный тип данных, подходящий для хранения любого размера в байтах.

Другие функции для выделения памяти (calloc, realloc) выходят за рамки данного курса.

Освобождение блока памяти

```
void free(void* ptr);
```

Функция освобождает блок памяти. Если `ptr` равен `NULL`, ничего не делает. Объявлена в заголовочном файле `stdlib.h`.

После вызова значение указателя `ptr` остается прежним, но разыменовывать его нельзя.

Неиспользуемую память нужно обязательно освобождать, иначе рано или поздно она может кончиться (утечка памяти).

Утечки памяти на серверных приложениях, которые должны работать несколько лет, недопустимы.

Утечки памяти в десктопных приложениях не так критичны, но могут и причинять неудобства (см. современные браузеры).

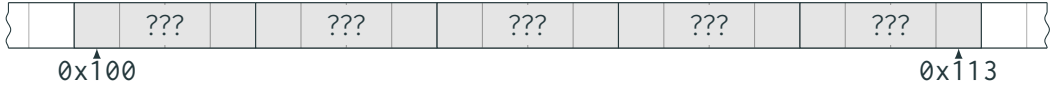
Пример: массив динамического размера

```
int n = read_number(); // 5
int* p; // ???
```

Выделение памяти под N целых чисел, где N задается во время исполнения программы.

sizeof — оператор, выдающий размер типов данных. Например, на Intel x86 sizeof(int) == 4.

Пример: массив динамического размера




```
int n = read_number(); // 5
int* p; // 0x100
p = malloc(n * sizeof(int));
if (p == NULL) { /* error */ }
```

Выделение памяти под N целых чисел, где N задается во время исполнения программы.

sizeof — оператор, выдающий размер типов данных. Например, на Intel x86 sizeof(int) == 4.

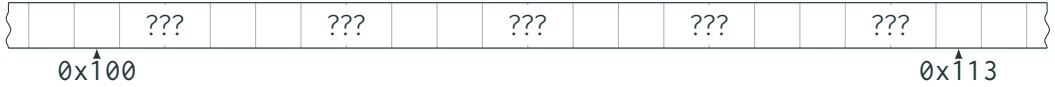
Пример: массив динамического размера


```
int n = read_number(); // 5
int* p; // 0x100
p = malloc(n * sizeof(int));
if (p == NULL) { /* error */ }
p[0] = p[n/2] = p[n-1] = 37;
```

Выделение памяти под N целых чисел, где N задается во время исполнения программы.

sizeof — оператор, выдающий размер типов данных. Например, на Intel x86 sizeof(int) == 4.

Пример: массив динамического размера




```
int n = read_number(); // 5
int* p; // 0x100
p = malloc(n * sizeof(int));
if (p == NULL) { /* error */ }
p[0] = p[n/2] = p[n-1] = 37;
free(p);
```

Выделение памяти под N целых чисел, где N задается во время исполнения программы.

sizeof — оператор, выдающий размер типов данных. Например, на Intel x86 sizeof(int) == 4.

Указатели на функции

Функции как данные

В языке C с функциями можно работать как с данными:

```
double (*my_func)(double, double) = pow;
double x = my_func(3, 2); // x = 9.0
```

Функция — это набор байтов в памяти, кодирующих тело этой функции с помощью машинных команд. Значит, можно взять адрес этого «набора байтов».

Более формально:

```
double (*my_func)(double, double) = &pow;
double x = (*my_func)(3, 2); // x = 9.0
```


pow — функция возведения в степень, объявлена в файле math.h.

Произносится следующим образом: переменная my_func — это указатель на функцию, принимающую два параметра типа double и возвращающую значение типа double.

Оператор взятия адреса & для функций можно опускать (чаще всего опускается).

Оператор разыменовывания * при вызове функции можно опускать (чаще всего опускается).

Заметим, что вы уже встречались с типами, окружающими имя переменной: int arr[10][5] — в данном случае переменная имеет имя arr и тип int[10][5].

Пример численного вычисления производной в точке

```c
double diff(double x, double (*f)(double)) {
    double dx = 0.01;
    return (f(x + dx) - f(x)) / dx;
}

double square(double x) {
    return x * x;
}

printf("%g\n", diff(M_PI/3, sin)); // 0.495662
printf("%g\n", diff(M_PI/6, cos)); // -0.504322
printf("%g\n", diff(3, square));   // 6.01
```


В данном примере с помощью указателей на функции мы можем численно вычислять производные математических функций, которые принимают один параметр типа double и возвращают результат типа double.

Заметим, что мы можем передавать параметром не только библиотечные функции (например, sin и cos из math.h), но и собственные функции (например, square).

M_PI — константа из math.h

Символы и кодировки

Символы в C

Символы задаются в одинарных кавычках, хранятся в переменных типа char.

```
char x = 'H';
char y = 'i';
printf("%c%c\n", x, y); // Hi
```

Кодировки

Кодировка — соответствие между символом и целочисленным кодом.

ASCII — фундаментальная 7-битная кодировка:

- 32 управляющих символа,
  - `'\0'`, код 0 — пустой символ,
  - `'\t'`, код 9 — табуляция,
  - `'\n'`, код 10 — перевод строки,
  - …
- 96 информационных символов.
  - `'0'..'9'` — цифры,
  - `'a'..'z'`, `'A'..'Z'` — буквы,
  - `','`, `'?'`, `...` — пунктуация.
  - …

То есть символы представляются просто своим целочисленным кодом.

American standard code for information interchange.

Табуляция — горизонтальный пробел для форматирования таблиц.

Символы как числа

```
char a = 'X';
printf("%c\n", a); // X
printf("%d\n", a); // 88

char b = a + 1;
printf("%c\n", b); // Y
```

Символ — одновременно и символ, и код. Все зависит лишь от способа его трактования.

Национальные символы

А что делать, если хочется использовать кириллицу, умляуты и иероглифы?

8-битные кириллические кодировки, базирующиеся на ASCII: КОИ-8, Windows-1251, ...

На практике как-то так: ишNºюърц §ыхъЋNºшєшърішц

КОИ-8 и Windows-1251 не совместимы друг с другом и не совместимы со всеми остальными национальными кодировками — бардак!

В браузерах до сих пор есть возможность вручную выбрать кодировку страницы. И это иногда бывает полезно.

Юникод

Юникод — многобайтовая кодировка, покрывающая почти все письменные языки.

Позволяет кодировать 1 112 064 символов. В версии 9.0, июнь 2016 г., используется лишь 128 237.

Содержит все национальные символы, математические символы, символы древних письменностей и многое другое, включая ☺ 😐 ☹ 😁 😂 😄 😇 😉 😈 😋 😍 😱 и даже 😹!

Юникод на данный момент используется по умолчанию во многих языках программирования, в операционных системах, на большинстве интернет сайтов.

Тонкости кодирования юникодных символов в виде байтов в данной лекции не рассматривается. Также не рассматриваются способы использования юникода в программах на C.

Интересная обзорная лекция про особенности юникода с TechTalks@NSU: https://techtalks.nsu.ru/36

Строки

Строки

Строка — это набор символов. В языке C строки представляются в виде массивов символов типа `char`. Константные строки задаются в двойных кавычках.

Длина строки явно не хранится: после последнего символа строки хранится специальный символ `'\0'`.

```
char* str = "Hi";
printf("%s\n", str);                  // Hi
printf("%d - %c\n", str[0], str[0]); // 72 - H
printf("%d - %c\n", str[1], str[1]); // 105 - i
printf("%d - %c\n", str[2], str[2]); // 0 -
```


Заметим, что для строки длины N необходим массив длины как минимум N+1.

Размер массива, в котором хранится строка ”Hi” — 3 байта.

Тонкость с инициализацией строк

- `test` — указатель на константную строку.

```
char* test = "cat";
test[2] = 'r'; // run time error
```

- `test` — массив, проинициализированный константной строкой.

```
char test[] = "cat";
test[2] = 'r';
printf("%s\n", test); // car
```

В первом случае переменная test указывается на специальную область памяти, доступную только для чтения, где хранятся все константные строки.

Во втором случае test указывает на массив в локальной памяти, доступной для чтения и записи, в который было скопировано содержимое константной строки.

Конец третьей лекции