OTUS

Онлайн образование

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Типы данных (продолжение)

Плисенко Ольга

Преподаватель курса «Программист С»

Эл. почта plolga.otus@yandex.ru
Telegram: @PlisenkoOlga

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в Telegram @OTUS C-2023-07

Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

Маршрут вебинара

- Массивы
- Строки
- Функции
- Файлы
- Рефлексия

Цели вебинара

После занятия вы сможете

1. Вспомнить классические типы данных
2. Создать массивы, строки, функции и файлы
3. Понять, как различные типы хранятся в памяти

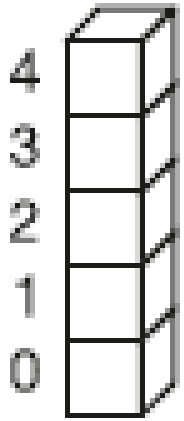
Что запомнилось с предыдущих вебинаров ???

1. Чем отличается локальная переменная от глобальной
2. Назовите классические типы данных
3. Перечислите пользовательские типы данных
4. В чем разница между структурой и объединением

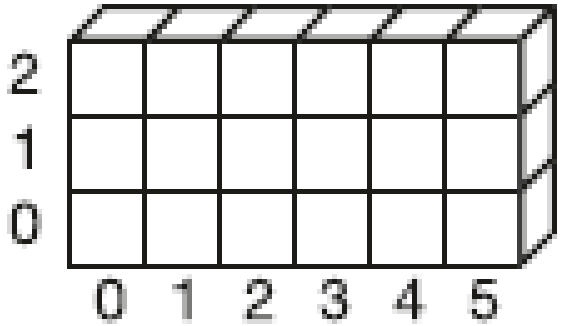
Массивы

Массив

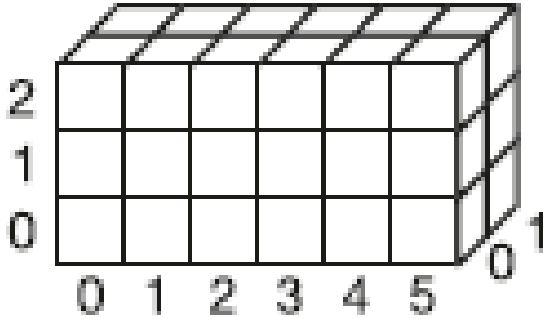
Массив – конечная последовательность однотипных величин, имеющая общее имя.



Одномерный
(вектор)



Двумерный
(матрица)



Трехмерный
(куб)

Массив

В зависимости от способа размещения в памяти, массивы бывают:

1. Статические – массивы, размер которых жестко определен в программе и не может меняться.

2. Динамические – массивы, размер которых может меняться во время исполнения программы.

Объявление массива

```
<тип>  <имя_массива>[<размер>];
```

Примеры объявления массива:

```
int A[4];
float B[50];
```

В стандарте C99+

```
double mas[5] = {50.0, 20.8, 42., 80.21, 41.69};
char str[] = "Hello";
```

Доступ к элементам массива

По индексу:

```
int B[20];
B[5] = 13;
B[10] = B[5] * 10;
```

Через указатель:

Раздел 6.5.2.1 Стандарта: E1[E2] <=> *(E1 + E2)

```
mas[2] == *(mas + 2) == *(2 + mas) == 2[mas]
&mas == mas
&p != p
```

Размер массива

```
<Размер> = sizeof(<базовый тип>)  * <количество_элементов>
```

следовательно:

```
<кол-во_эл-ов> = sizeof (mas) / sizeof (int);
<кол-во_эл-ов> = sizeof (mas) / sizeof (mas[0]);
```

Многомерные массивы

```
int a[3][4] = {
    {0, 1, 2, 3} ,
    {4, 5, 6, 7} ,
    {8, 9, 10, 11}
};
```

Row-major order:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

```
float a[2][2] = {{3.2, 4.3}, {1.1, 2.2}};
char sym[2][2] = { 'a', 'b', 'c', 'd' };
```

Массив структур

Массив структур:

```
struct student otus[30];
otus[i].number
```

Динамический массив структур:

```
struct student *otus;
otus = (struct student*) malloc (count_rec * sizeof(struct student));
otus->number
(*otus).number
```

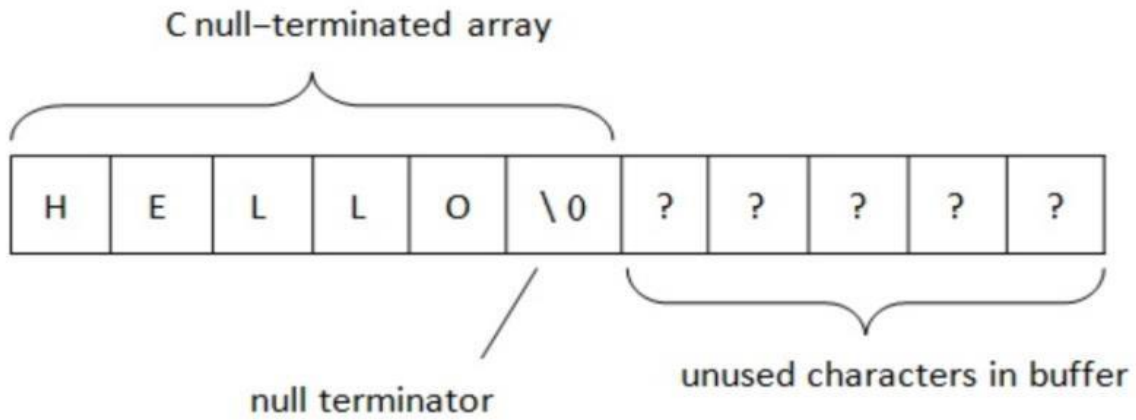
Итого по массивам

1. Переменные массивов могут быть использованы в качестве указателей…
2. … но переменные массивов и указатели – это НЕ одно и тоже.
3. sizeof возвращает разные значения для указателей и переменных массивов.
4. Переменные массивов НЕ могут указывать на что-либо другое.
5. Если присвоить указателю переменную массива, то она теряет информацию о количестве элементов (распад массива).

Строки

Объявление строк

```
char str1[20] = "Строка";   // один байт под \0
char str2[10] = {'S', 't', 'r', 'i', 'n', 'g', '\0'};
char str3[] = "HELLO";
char *str4 = "это тоже строка";
char *str5 = "very very"
             "very very"
             "long string";
```



Выделение памяти под строку

```
char *str = (char *) malloc (sizeof(char) * strlen(buffer));
```

что правильнее ???

```
char *str = (char *) malloc (sizeof(char) * (strlen(buffer) +1));
```

Функции работы со строками

1. Ввод и вывод строк.
2. Преобразование строк.
3. Обработка строк.

Ввод/вывод строк

1. Ввод строк:
   - scanf(“%s”, str); - до первого пробела
   - gets(str); - не учитывает длину
   - fgets(str, длина, stdin); - желательно очищать поток вода fflush(stdin)
2. Вывод строк:
   - printf(“%s”, str);
   - puts(str); - начинает вывод с новой строки
   - fputs(str, stdout); - вывод с текущего положения курсора

Преобразование строк

1. `atof(const char *string); // преобразование строки в число типа double`
2. `int atoi(const char *string); // преобразование строки в число типа int`
3. `long int atol(const char *string); // преобразование строки в число типа long int`
4. `long long int atoll(const char *string); // преобразование строки в число типа long long`

Обработка строк

Библиотека string.h))

Дополнительная информация

Алгоритм маляра Шлемиля (Спольски Дж. Джоэл о программировании).

Poul-Henning Kamp, The Most Expensive One-byte Mistake: https://queue.acm.org/detail.cfm?id=2010365

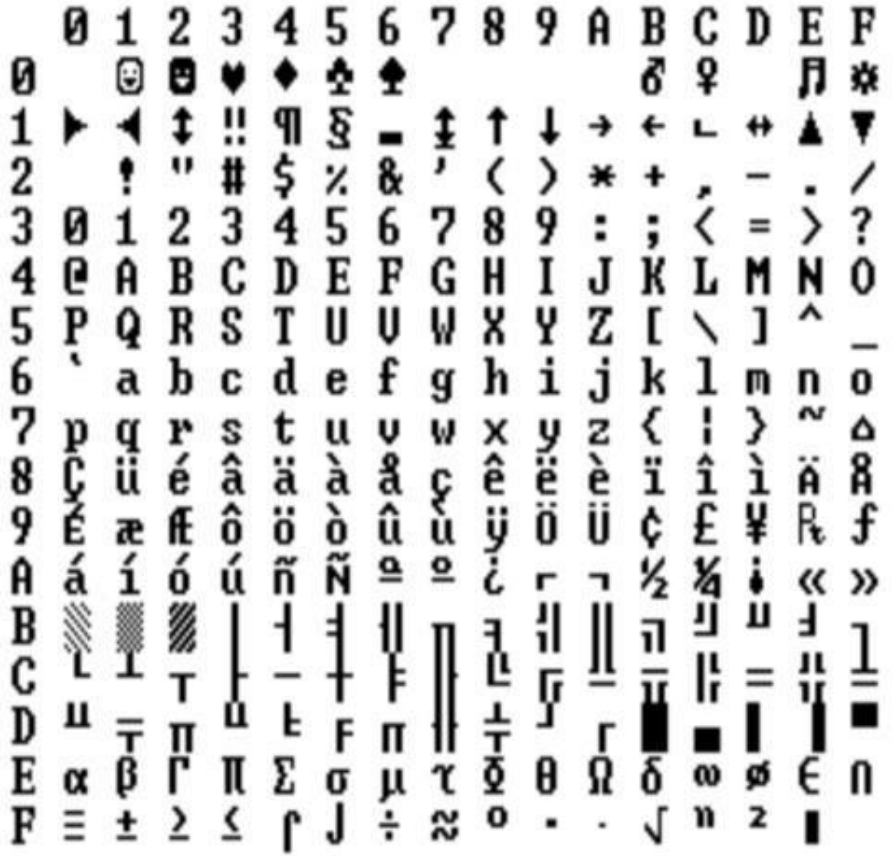
Список строковых функций: https://en.wikipedia.org/wiki/C_string_handling

C gibberish <-> english: https://cdecl.org

Абсолютный минимум, который каждый разработчик должен знать о Unicode и кодировках: https://cyberforum.ru/blogs/33029/blog5139.html

Unicode

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SPC	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL



Unicode

• Миф: Unicode - это просто 16-ти битный код, где каждый символ занимает 16 бит и содержит 65536 возможных символов.

• Code points. Cyrillic Capital Letter A - U+0410.

• Кодировки: UTF-8, UTF-16 (LE/BE), UTF-7, UCS-4 и т.д.

• Нет смысла держать строку, не зная в какой она кодировке.

• Unicode sandwitch

Библиотеки для обработки Unicode

• GLib - низкоуровневая библиотека, расширяющая возможности, предоставляемые стандартной библиотекой libc языка C, содержит обширный набор вспомогательных функций для строк и типовых структур данных.

• libicu (International Components for Unicode - ICU) – библиотека с открытым исходным кодом для C / C ++ и Java, обеспечивающая поддержку Unicode и интернационализации программного обеспечения. ICU - проект технического комитета Консорциума Unicode и спонсируется, поддерживается и используется IBM и многими другими компаниями.

• utf8.h (header-only) - заголовочный файл для поддержки строк utf8 в C и C++. Содержит как функции, предоставляемые из библиотеки strings.h, но с префиксом utf8* вместо префикса str*, так и дополнительные функции.

• GNU libunistring - библиотека реализует строковые типы Unicode в трех вариантах: (UTF-8, UTF-16, UTF-32) вместе с функциями для обработки символов (имена, классификации, свойства) и функциями для обработки строк (итерация, форматированный вывод, ширина, разрывы слов, разрывы строк, нормализация, сворачивание регистра и регулярные выражения).

Unicode

```c
#include <iconv.h>
int main()
{
    const char* in = "Вопрос!";  char out[1024];
    char* in_ptr = in;
    char* out_ptr = &out[0];

    size_t inbytes = strlen(in);
    size_t outbytes = sizeof out / sizeof out[0];

    iconv_t cd = iconv_open("koi8-r", "utf-8");
    iconv(cd, &in_ptr, &inbytes, &out_ptr, &outbytes);
    iconv_close(cd);
    printf("%s\n", out);

    return 0;
}
```

Функции

Функции

С не является функциональным ЯП.

```
<тип> <имя_функции> (<список параметров>)
{
    <тело функции>;
}
```

Функция может возвращать значения любого типа , кроме массива (раздел 6.9.1.3 Стандарта)

Функции

Передача параметров:
по ссылке (call by reference) – передается адрес переменной
по значению (call by value) – передается копия

Аргументы функции main()
int argc – количество аргументов командной строки (мин. 1 – имя программы)
char *argv[] – массив, в который заносятся все аргументы командной строки
char *envp[] – массив указателей на переменные среды (расширение Microsoft для ANSI C)

```c
int main(int argc, char *argv[])
{
    char *path;

    if (argc !=2)   {
        printf ("File name not specified!");
        return EXIT_FAILURE;
    }
    path=argv[1];
```

Передача структур в функцию

1. Передача частей структуры в функцию:

```
f(otus.number);
f(otus.code);
f(otus.code[3]);
```

2.Передача структуры в качестве аргумента (!!! Структура копируется в стек функции !!!)

```
int f_num(struct student o)
....
f_num(otus);
```

3. Передача структуры через указатель

```
int f_num(struct student *o)
....
f_num(&otus);
```

Функции

В стандарте C89 указывать return – не обязательно

Со стандарта C99 указывать return – обязательно

Файлы

Файлы

FILE* - т.н. opaque pointer (“чеширский кот”)

```c
#include <stdio.h>
FILE* stdin;
FILE* stdout;
FILE* stderr;
FILE* fopen(const char* filename, const char* mode);
// mode: "r", "w", "a"; modifiers: "+", "b", "x"
int fclose(FILE* fp);
int fflush(FILE* fp);
```

Чтение из файла

```
int fgetc(FILE* fp);

char* fgets(char* str, int num, FILE* fp);

size_t fread(void* ptr, size_t size, size_t count, FILE* fp);

int fscanf(FILE* fp, const char* format, ...);
```

Запись в файл

```c
int fputc(int character, FILE* fp);

int fputs(const char* str, FILE* fp);

size_t fwrite(const void* ptr, size_t size, size_t count, FILE* fp);

int fprintf(FILE* fp, const char* format, ...);
```

Перемещение указателя в файле

Поддержка операций ввода/ вывода по произвольному адресу:

```
int fseek(FILE* fp, long int offset, int origin);
```

- устанавливает указатель положения в файле в соответствии со значениями offset и origin:

  offset — это выраженный в байтах сдвиг от позиции, определяемой origin, до новой позиции;

  origin может принимать значения : SEEK_SET или 0 (начало файла), SEEK_CUR или 1 (текущая позиция), SEEK_END или 2 (конец файла)

- возвращает 0 в случае успеха.

Рефлексия

Рефлексия

Что запомнилось с вебинара?

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Плисенко Ольга

Эл. почта plolga.otus@yandex.ru
Telegram: @PlisenkoOlga