СТРОКИ И ФАЙЛЫ

С-строки. Некоторые интересные алгоритмы на строках. Работа с файловой системой

К. Владимиров, Intel, 2018
mail-to: konstantin.vladimirov@gmail.com

Значения символов

- То, что вы привыкли считать символами, на самом деле кодируется числами

```
int c = 'a';

printf("%d\n", c); // что на экране?
```

- И наоборот достаточно небольшие цифры это коды для символов

```
char x = 58;

printf("%c\n", x); // а сейчас?
```

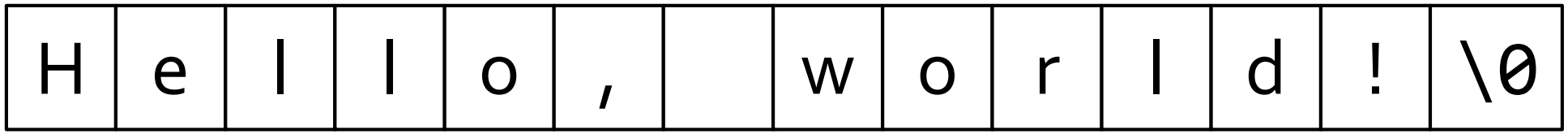
- В Unix формат символов это UTF-8. Его первые 127 символов совпадают с таблицей ASCII
- Упражнение: распечатайте символы с 1 по 127 рядами по 16

Строковые литералы

- Следующая строчка может показаться смутно знакомой

```
printf("%s\n", "Hello, world!");
```

- Символы `"Hello, world"` в кавычках это строковый литерал



- Любая строка в языке C оканчивается завершающим нулевым символом
- Нулевой символ `'\0'` это не символ `'0'`. Это спецсимвол, имеющий код `0`.

Изменяемые строки

- Самый простой способ создать изменяемую строку это объявить массив символов

```
char hello[20] = "Hello, world!";
```

- Символы после завершающего нуля (он тут 14-й) содержат мусор
- Далее содержимое это строки можно поменять

```
hello[1] = 'a';

printf("%s\n", hello); // → Hallo, world!
```

- Изменяемая строка это любой массив символов, завершающийся нулём
- Символы после завершающего нуля ничего не значат

Копирование строк

- Одна строка может быть скопирована в другую

```
char hello[20] = "Hello, world!";
char duplicate[20];

const char *src = hello;
char *dst = duplicate;
while (*src != '\0') {
  *dst = *src; dst += 1; src += 1;
}
*dst = '\0'; // не забыть установить завершающий ноль
```

- В библиотеке есть специальная функция strcpy

```
strcpy (duplicate, hello); // то же самое
```

Копирование строк

- Одна строка может быть скопирована в другую

```
char hello[20] = "Hello, world!";
char duplicate[20];

const char *src = hello;
char *dst = duplicate;

// немного джигитовки в стиле C
while ((*dst++ = *src++) != '\0') {}
```

- В библиотеке есть специальная функция strcpy

```
strcpy (duplicate, hello); // то же самое
```

Работа с C-строками

- Напишите самостоятельно (на время 2.5 балла за каждую)

```
char *strcat (char *dst, const char *src) {
  // дописать src в конец dst
}

int strlen(const char *src) {
  // вычислить длину строки
}

int strcmp(const char *lhs, const char *rhs) {
  // Сравнить посимвольно в словарном порядке. Вернуть -1, 0 или 1
}

const char *strchr(const char *src, int c) {
  // определить позицию символа в строке
}
```

Проблема Шлемиля-маляра

- Кто видит проблемы в этом коде?

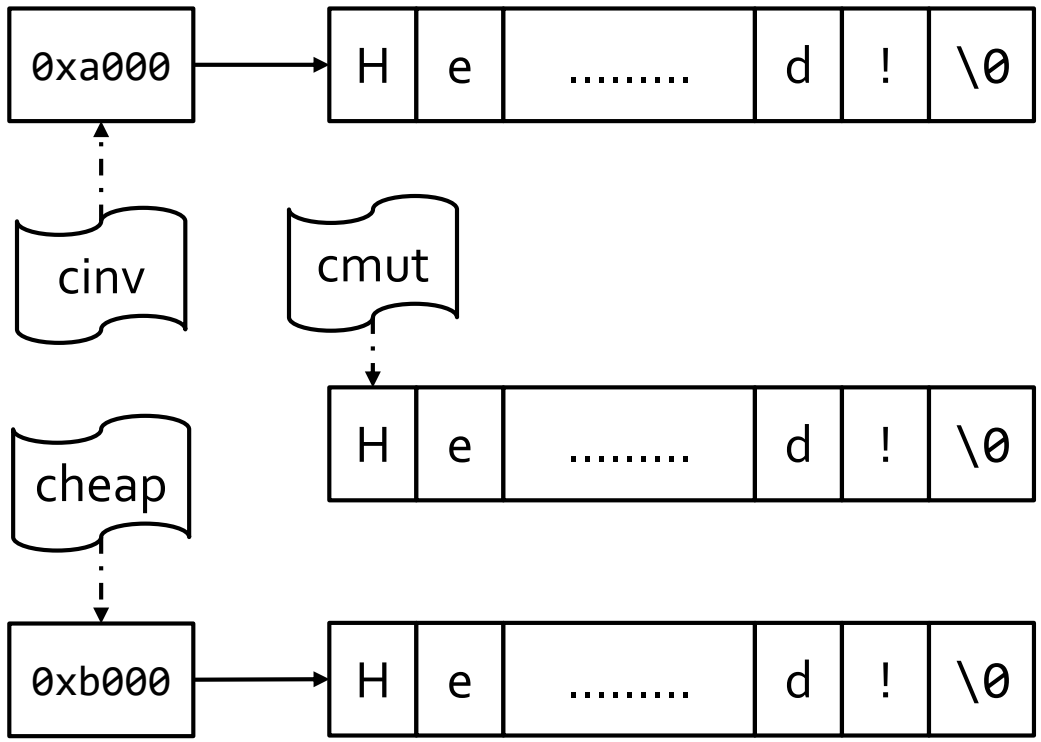
```
char buf[MAXSZ];
strcpy(buf, "First ");
strcat(buf, "Second ");
strcat(buf, "Third ");
strcat(buf, "Fourth ");
```

- Как бы вы переписали этот код?

C-строки и разные виды памяти

- Обычно используют указатель на первый элемент

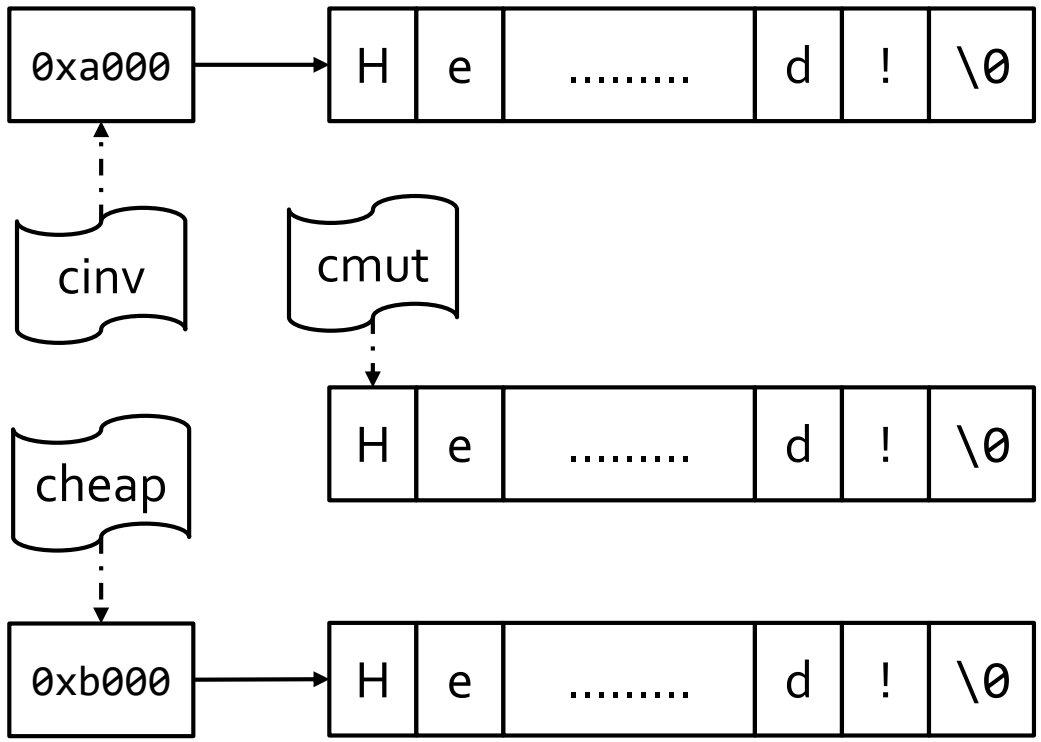
```
const char *cinv = "Hello, world";
char cmut[] = "Hello, world";
char *cheap = (char *) malloc (50);
strcpy (cheap, cinv);
cheap = cinv;
cinv = 0;
cmut = cheap;
```



C-строки и разные виды памяти

- Обычно используют указатель на первый элемент

```
const char *cinv = "Hello, world";
char cmut[] = "Hello, world";
char *cheap = (char *) malloc (50);
strcpy (cheap, cinv);
cheap = cinv; // утечка памяти
cinv = 0; // ok
cmut = cheap; // ошибка компиляции
```



Problem SR: переворот строки

- Напишите функцию strrev которая переворачивает строчку на месте

```
char *strrev(char *str) {
  // TODO: write code
}
```

- Например

```
char *buf = (char *) calloc(50, sizeof(char));
strcpy(buf, "1 23 456 78 9");
strrev(buf);
assert(0 == strcmp(buf, "9 87 654 32 1");
```

- При переворачивании не должно выделяться дополнительной памяти
- Не забудьте про завершающий ноль!

Problem SU: поиск подстроки

- Напишите простую функцию `strstr`, которая ищет подстроку в строке

```
const char *strstr(const char *haystack, const char *needle);
```

- Например

```
const char *needle = "ab";
const char *src = "abracadabra";
const char *pos1, *pos2, *pos3;
pos1 = strstr(src, needle); assert(pos1 != NULL);
pos2 = strstr(pos1 + 2, needle); assert(pos2 != NULL);
pos3 = strstr(pos2 + 2, needle); assert(pos3 == NULL);
```

- Первая найденная позиция "abracadabra", вторая "abracadabra"
- Оцените асимптотику наивного алгоритма

HWK: алгоритм КМП (optional)

- Более интересный алгоритм Кнута-Морриса-Пратта делает из строки автомат, вычисляя так называемую failure function `f[i]`

W[i]	A	B	A	C	A	B
S[0:i]	A	AB	ABA	ABAC	ABACA	ABACAB
f[i]	0	0	1	0	1	2

- Далее с использованием этой табличной функции идёт поиск

1	2	3																		
A	B	A	B	A	C	A	A	B	A	C	A	B	C	A	C	B	C	A	A	A
A	B	A	C	A	B															

- Смотрим `f[2] == 1`, сдвигаем на `3 - f[2] == 2`

HWK: алгоритм КМП (optional)

- Более интересный алгоритм Кнута-Морриса-Пратта делает из строки автомат, вычисляя так называемую failure function `f[i]`

W[i]	A	B	A	C	A	B
S[0:i]	A	AB	ABA	ABAC	ABACA	ABACAB
f[i]	0	0	1	0	1	2

- Далее с использованием этой табличной функции идёт поиск

		1	2	3	4															
A	B	A	B	A	C	A	A	B	A	C	A	B	C	A	C	B	C	A	A	A
		A	B	A	C	A	B													

- Смотрим `f[3] == 0`, сдвигаем на `4 - f[3] == 4`

HWK: алгоритм КМП (optional)

- Более интересный алгоритм Кнута-Морриса-Пратта делает из строки автомат, вычисляя так называемую failure function `f[i]`

W[i]	A	B	A	C	A	B
S[0:i]	A	AB	ABA	ABAC	ABACA	ABACAB
f[i]	0	0	1	0	1	2

- Далее с использованием этой табличной функции идёт поиск

						1														
A	B	A	B	A	C	A	A	B	A	C	A	B	C	A	C	B	C	A	A	A
						A	B	A	C	A	B									

- Смотрим `f[0] == 0`, сдвигаем на `1 - f[0] == 1`

HWK: алгоритм КМП (optional)

- Более интересный алгоритм Кнута-Морриса-Пратта делает из строки автомат, вычисляя так называемую failure function `f[i]`

W[i]	A	B	A	C	A	B
S[0:i]	A	AB	ABA	ABAC	ABACA	ABACAB
f[i]	0	0	1	0	1	2

- Далее с использованием этой табличной функции идёт поиск

							m	a	t	c	h	!								
A	B	A	B	A	C	A	A	B	A	C	A	B	C	A	C	B	C	A	A	A
							A	B	A	C	A	B								

- Нашли подстроку. **Дома: реализуйте этот алгоритм на языке C**

Расстояние редактирования

- Как перейти от слова spoon к слову sponge?
- Что можно делать:
  - Вставлять букву в любое место (цена == 1)
  - Удалять любую букву (цена == 1)
  - Изменять любую букву (цена == 2)
- Нужно найти последовательность действий с минимальной ценой

spoon → sponn (+2) → spong (+2) → sponge (+1), cost = 5

spoon → spon (+1) → spong (+1) → sponge (+1), cost = 3

Динамическое программирование

- Любая часть оптимальной траектории является оптимальной траекторией

$$D(i,j) = min\begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1)+K(i,j) \end{cases}$$

$$K(i,j) = \begin{cases} 0, & S_1(i) = S_2(j) \\ 2, & S_1(i) \neq S_2(j) \end{cases}$$

- Задача теперь сводится к тому, чтобы найти $D(5,6)$
- Потренируемся заполнять таблицу

	#	S	P	O	N	G	E
#	0	1	2	3	4	5	6
S	1	$D(1,1)$					
P	2						
O	3						
O	4						
N	5						$D(5,6)$

Problem ED: расстояние редактирования

- Реализуйте алгоритм расстояния редактирования как функцию на C

```
struct costs {
  int addcost, removecost, replacecost;
};

int edistance (const char *fst, const char *snd, struct costs c);
```

- Найдите расстояние между следующими строчками для цен `{1, 1, 2}`

AGGCTATCACCTGACCTCCAGGCCGATGCCC

TAGCTATCACGACCGCGGTCGATTTGCCCGAC

- Как вы будете тестировать этот алгоритм?

Реаллокации

- При работе со строкой иногда хочется расширить буфер в динамической памяти, с которым идёт работа
- Здесь на помощь приходит `realloc`

```
void *realloc (void *ptr, size_t new_size);
```

- Например

```
int *arr = (int *) malloc(100);
arr = (int *) realloc (arr, 1000);
```

- При нехватке памяти, `realloc` возвращает `NULL`
- Внимание: `realloc` не работает на стеке и в глобальной памяти

Problem SP: замена в строке

- Реализуйте функцию, осуществляющую замену всех подстрок в строке
- Функция должна аллоцировать и возвращать новую строку

```
char *replace(const char *str, const char *from, const char *to);
```

- Например

```
char *s = replace("Hello, %username, how are you, %username",
                  "%username", "Eric, the Blood Axe");
printf("%s\n", s);
free(s);
```

- Обратите внимание, что строка для замены может быть и длиннее и короче заменяемой

Обсуждение

- Как ещё могли бы быть устроены строки?
- Являются ли C-строки лучшей альтернативой?

Аргументы функции main

- До сих пор мы пользовались main() или main(void)
- Но есть вторая форма основной функции

```
int main(int argc, char **argv) {
  // тело функции
}
```

- `argc` это количество аргументов
- `argv` это массив `argc+1` строк
- `argv[0]` это имя самой программы

Problem EA: вывести аргументы

- Напишите программу которая будет печатать аргументы с которыми вызвана, заменяя `'-'` на `'+'`

```
> ./a.out -a t -x --b 123
-a t +x ++b 123
```

- Если среди этих аргументов есть `-r`, программа должна вывести их в обратном порядке

```
> ./a.out -a t -r -x --b 123
123 ++b +x +r t +a
```

- Как вы думаете почему именно `'-'` является стандартным для аргументов командной строки?

Минимум о работе с файлами

- Файл (`FILE`) это синоним для implementation defined структуры

```
typedef struct _File_t { /* нечто * / } FILE;
```

- Даже sizeof(FILE) не определён. Мы всегда оперируем с FILE*
- Открыть файл можно с помощью функции `fopen`

```
FILE *fopen (const char *name, const char *mode); // sets errno
```

- Базовые режимы открытия: "`r`", "`w`", "`a`" для текстовых и "`rb`", "`wb`", "`ab`" для бинарных. Означают запись, перезапись и дозапись в конец
- Закрыть файл можно с помощью функции `fclose`

```
int fclose (FILE *stream);
```

Минимум о работе с файлами

- Ввод и вывод бывают форматированными и не форматированными
- Форматированный ввод/вывод это `fprintf` и `fscanf`

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

- Форматные модификаторы все как в обычных `printf` и `scanf`
- Неформатированный ввод и вывод это посимвольное считывание

```
int fputc (int character, FILE *stream);
int fputs (const char *str, FILE *stream);
int fgetc (FILE *stream);
```

- Специальные файлы: `stdin`, `stdout`, `stderr` – все тоже имеют тип `FILE*`

Problem FO: вывод файла на экран

- Создайте произвольный текстовый файл на жёстком диске
- Пример содержимого файла `myfile.txt`

```
quick brown fox jumps over lazy dog
```

- Напишите программу, которая принимает его имя как аргумент, считывает его и выводит на экран как текстовый файл

```
> gcc outtext.c -o outtext
> ./outtext myfile.txt
quick brown fox jumps over lazy dog
```

Problem FO2: вывод бинарного файла

- Создайте произвольный текстовый файл на жёстком диске
- Пример файла `myfile.in`

```
1234
```

- Напишите программу, которая выводит файл, являющийся её аргументом `argv[1]` на экран как последовательность 16-ричных чисел

```
> gcc outbin.c -o outbin
> ./outbin myfile.in
0x30 0x31 0x32 0x33
```

- Выведите саму эту программу с помощью неё же

```
> ./outbin a.out
```

Перемещение внутри файла

- Для навигации внутри файла используются

```
long int ftell (FILE *stream); // текущее положение

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

SEEK_SET	Начало файла
SEEK_CUR	Текущее положение в файле
SEEK_END	Конец файла

```
pFile = fopen ("example.txt", "wb"); // бинарная запись
fputs ("This is an apple", pFile);
fseek (pFile, 9, SEEK_SET);
fputs (" sam", pFile); // → This is a sample
```

Проблем FS: поиск внутри файла

- Необходимо написать функцию, которая принимая в себя имя файла, начальную позицию и строчку, возвращает первую найденную позицию этой строки в файле (в символах от начала файла) или `-1` если строчка не была найдена

```
int strstrf (const char *name, int offset, const char *substr);
```

- Решая эту задачу, предположите, что файл гигантский и полностью считать его в память нельзя
- Можно воспользоваться любой из стандартных функций из `<string.h>`

Проблем SF: сортировка файла

- Необходимо написать функцию, которая принимая в себя имя текстового файла и сортировала слова в нём, выводя результат в другой файл

```
int sortf (const char *in, const char *out);
```

- Например входной файл:

```
quick brown fox jumps over lazy dog
```

- Выходной файл

```
brown dog fox jumps lazy over quick
```

- Решая эту задачу, предположите, что файл гигантский и полностью считать его в память нельзя, но вы можете создавать временные файлы
- В файле все слова разделены пробелами и переносом строк

Литература

- [C11] ISO/IEC – "Information technology – Programming languages – C", 2011
- [$K\&R$] Brian W. Kernighan, Dennis Ritchie – The C programming language, 1988
- [JSP] Joel Spolsky – Back to Basics, 2001
- [$Linden$] Peter van der Linden – Expert C Programming: Deep C Secrets , 1994
- [$Cormen$] Thomas H. Cormen – Introduction to Algorithms, 2009
- [$TAOCP$] Donald E. Knuth – The Art of Computer Programming, 2011
- [$SALG$] Robert Sedgewick Algorithms, 4th edition, 2011