



РОБОТА З РЯДКАМИ В МОВІ ПРОГРАМУВАННЯ C

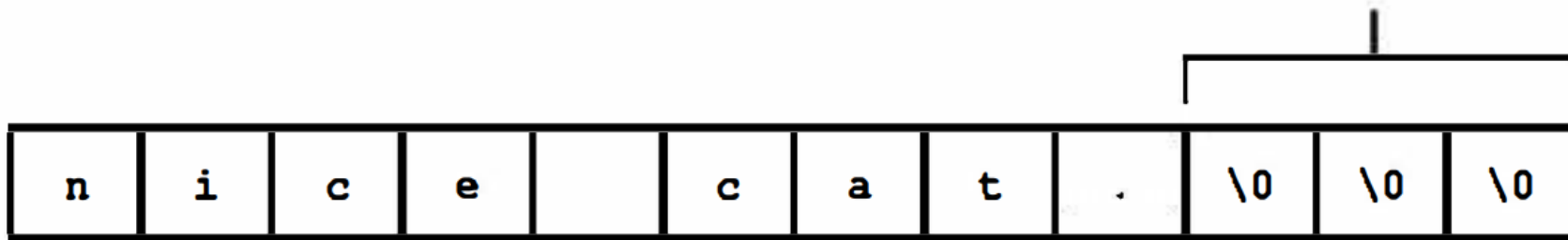
Питання 4.2.

Рядки як символьні масиви

- Символьний рядок являє собою масив значень типу `char`, кінцевим елементом якого є нульовий символ (`\0`).
 - У зв'язку з інтенсивним використанням символьних рядків мова С надає багато функцій, що призначені для роботи з рядками.
- Визначення рядків передбачає використання:
 - рядкових констант (рядкових літералів):
 - "Здравствуйте, ну и как вы себя чувствуете сегодня?"
 - масивів символів типу `char`:
 - `const char m1[40] = "Постарайтесь уложиться в одну строку.";`
 - Скорочено від `const char m1 [4 0] = { ' П ' , ' о ' , ' с ' , ' т ' , ' а ' , ' р ' , ' а ' , ' й ' , ' т ' , ' е ' , ' с ' , ' Ъ ' , ' , ' у ' , ' л ' , ' о ' , ' ж ' , ' и ' , ' т ' , ' Ъ ' , ' с ' , ' я ' , ' , ' в ' , ' , ' о ' , ' д ' , ' н ' , ' у ' , ' , ' с ' , ' т ' , ' р ' , ' о ' , ' к ' , ' у ' , ' , ' \0 ' };`
 - Константы типа символьной строки размещаются в классе статической памяти.
 - Вказівників типу `char`
 - `const char *m3 = " \ nВсе, о себе достаточно , а вас как зовут?";`

Ініціалізація рядка у вигляді масиву

Надмірні елементи ініціалізуються символом \0



```
char pets[12] = "nice cat.";
```

- Ініціалізація символьних масивів – один з випадків, коли задачу визначення розмірів можна покласти на компілятор.
 - `const char m2[] = "Если вам ничего не приходит в голову, придумайте что-нибудь .";`
 - Розмір масиву фіксується в програмі під час її компіляції.
 - Загалом у стандарті C99 можна використовувати масиви змінної довжини, проте і в такому випадку потрібно знати, наскільки великим він має бути.

Масиви символьних рядків

```
const char *mytal[LIM] = { "Мгновенное складывание чисел",  
    "Точное умножение", "Накапливание данных",  
    "Исполнение инструкций с точностью до последней буквы",  
    "Знание языка программирования C" };
```

- Можна стверджувати, що `mytal` – це масив, який складається з 5 вказівників на значення типу `char`.
 - `mytal[0]` вказує на перший символ першого рядка: `*mytal[0] == 'M'`. Аналогічна конструкція – `mytal[0][0]`.
 - `*mytal[1] == 'T'`.
- Інший підхід – створити двовимірний масив: `char mytal_2[LIM][LINLIM]`;
 - Прямокутний масив, у кожному рядку 81 елемент.
 - Проте масив вказівників утворює масив записів різної довжини та не потребує зайвої пам'яті.
- `mytal` та `mytal_2` мають різні типи:
 - `mytal` – масив вказівників на значення типу `char`
 - `mytal_2` – масив значень типу `char`.
 - `mytal` містить 5 адрес, а `mytal_2` - 5 повних символьних масивів.

Масиви символних рядків

A	p	p	l	e	\0	\0
---	---	---	---	---	----	----

P	e	a	r	\0	\0	\0
---	---	---	---	----	----	----

O	r	a	n	g	e	\0
---	---	---	---	---	---	----

```
char fruit[3][7]=  
{ "Apple",  
  "Pear",  
  "Orange"  
};
```

A	p	p	l	e	\0
---	---	---	---	---	----

P	e	a	r	\0
---	---	---	---	----

O	r	a	n	g	e	\0
---	---	---	---	---	---	----

```
char* fruit[3]=  
{ "Apple",  
  "Pear",  
  "Orange"  
};
```

Различия
в объявлениях

Рядки та вказівники

- Більша частина операцій над рядками в мові С фактично використовує вказівники.

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

```
int main(void)
```

```
{
```

```
    const char * mesg = "Don't be a fool!";
```

```
    const char * copy;
```

```
    copy = mesg;
```

```
    printf("%s\n", copy);
```

```
    printf("mesg = %s; &mesg = %p; value = %p\n", mesg, &mesg, mesg);
```

```
    printf("copy = %s; &copy = %p; value = %p\n", copy, &copy, copy);
```

```
    return 0;
```

```
}
```

Сам рядок не копіювався жодного разу.

Оператор copy = mesg; лише створює другий вказівник, що посилається на той же рядок.

C:\Users\User\AppData\Local\Temp\Rar\$Dla1680.10712\p_and_s.exe

```
Don't be a fool!
```

```
mesg = Don't be a fool!; &mesg = 000000000064FE58; value = 0000000000409020
```

```
copy = Don't be a fool!; &copy = 000000000064FE50; value = 0000000000409020
```

Ввід рядків

- Потрібно
 - 1) зарезервувати простір пам'яті для зберігання рядка;
 - 2) використати функцію вводу для зчитування рядка.
- Нехай маємо код:
`char *name;`
`scanf ("%s", name);`
 - Компілятор не знайде помилок, проте при зчитуванні імені воно може записатись понад даними чи кодом програми, що призведе до аварійного завершення.
 - Функція `scanf()` копіює інформацію за адресою, заданою аргументом (тут – неініціалізований вказівник); `name` може вказувати на будь-яке місце в пам'яті.
- Найпростіше вирішення – включити явно заданий розмір масиву в оголошення:
 - `char name[81];`
 - Інша можливість – використання функцій розподілу пам'яті з бібліотеки C (питання 3).

Ввід рядків. Функція gets() з stdio.h

- Бібліотека C надає 3 функції для зчитування рядків: scanf(), gets() та fgets().
 - Частіше використовується gets().
 - Отримує рядок від стандартного пристрою вводу-виводу системи (поки не введено \n).

```
/* name1.c -- программа считывает имя */
#include <stdio.h>
#define MAX 81
int main(void)
{
    char name[MAX];          /* выделить пространство памяти */
    printf("Как вас зовут?\n");
    gets(name);              /* поместить строку в массив name */
    printf("Прекрасное имя, %s.\n", name);
    return 0;
}
```

Как вас зовут?

Васисуалий Лоханкин

Прекрасное имя, Васисуалий Лоханкин

Функція gets() з stdio.h

```
/* name2.c -- программа считывает имя */
#include <stdio.h>
#define MAX 81
int main(void)
{
    char name[MAX];
    char * ptr;
    printf("Как вас зовут?\n");
    ptr = gets(name);
    printf("%s? А! %s!\n", name, ptr);
    return 0;
}
```

■ Отримує ввід 2ма способами:

- 1) використовує адресу для завантаження рядка в масив name.
- 2) у кодї функції gets() використовується ключове слово return для повернення адреси рядка, а програма присвоює цю адресу вказівнику ptr.
 - ptr є вказівником на char .
 - функція gets() повинна повернути значення, яке являтиме собою вказівник на char.

Как вас зовут?

Алибаба Сорокаразбойников

Алибаба Сорокаразбойников? А! Алибаба Сорокаразбойников!

Функція fgets()

- `gets()` не виконує перевірку того, чи вміщаються вхідні дані в зарезервовану область пам'яті.
 - При переповненні зайві символи потрапляють в сусідні області пам'яті.
 - Функція `fgets()` виправляє таку поведінку, дозволяючи задати верхню межу кількості зчитаних символів.
- Відмінності `fgets()` від `gets()`:
 - Другий аргумент задає максимальну кількість символів для зчитування.
 - Для значення `n` виконається зчитування `n-1` символів або зчитування до наступного символу нового рядка залежно від того, що відбудеться першим.
 - Якщо функція `fgets()` стикається з символом нового рядка, вона зберігає його в рядку, на відміну від функції `gets()`, яка відкидає його.
 - Третій аргумент показує, з якого файлу повинно виконуватись зчитування.
 - Для зчитування з клавіатури використовується ідентифікатор `stdin` з `stdio.h`.

Функція fgets()

```
#include <stdio.h>
#define MAX 81
int main(void)
{
    char name[MAX];
    char * ptr;
    printf("Как вас зовут?\n");
    ptr = fgets(name, MAX, stdin);
    printf("%s? А! %s!\n", name, ptr);
    return 0;
}
```

```
Как вас зовут?
Киса Воробьянинов
Киса Воробьянинов
? А! Киса Воробьянинов
!
```

- Для видалення символу нового рядка можна використовувати функцію strchr().
- Функція gets() вважається ненадійною.
 - Для критичних програм потрібно використовувати функцію fgets()

Функція scanf()

- Функції scanf() і gets() по-різному визначають досягнення кінця рядка: scanf() більше орієнтована на «отримання слова», а не на «отримання рядка».
 - Функція scanf() може застосувати 2 варіанти для припинення вводу.
 - Для формату %s рядок продовжується до наступного пробільного символу (невключно).
 - При заданій ширині поля, наприклад %10s, функція scanf() зчитує до 10 символів або всі символи до першого пробільного символу.
 - Функція scanf() повертає цілочисельне значення, рівне кількості елементів послідовно зчитаних символів або символ EOF, якщо функція стикається з кінцем файлу.

Оператор вводу	Исходная очередь ввода *	Содержимое строки имени	Остальная часть очереди
scanf("%s", name);	Fleebert Hup	Fleebert	Hup
scanf("%5s", name);	Fleebert Hup	Fleeb	Ert Hup
scanf("%5s", name);	Ann Ular	Ann	Ular

Функція scanf()

```
#include <stdio.h>
int main(void)
{
    char name1[11], name2[11];
    int count;

    printf("Введіть, будь ласка, два імена.\n");
    count = scanf("%5s %10s", name1, name2);
    printf("Прочитано %d імен: %s і %s.\n",
           count, name1, name2);

    return 0;
}
```

```
Введіть, будь ласка, 2 імена.
Jesse Jukes
Прочитано 2 імен: Jesse и Jukes.

Введіть, будь ласка, 2 імена.
Liza Applebottham
Прочитано 2 імен: Liza и Applebott.

Введіть, будь ласка, 2 імена.
Portensia Callowit
Прочитано 2 імен: Porte и nsia.
```

- У першому прикладі обидва імена потрапляють в допустимі межі розмірів.
 - У другому прикладі були зчитані тільки перші 10 символів імені Applebottham через використання формату %10s.
 - У третьому прикладі 4 букви імені Portensia потрапляють в name2, оскільки другий виклик scanf() відновлює ввід там, де закінчується перший ввід (тут – всередині слова Portensia).

Виведення рядків

- У мові C доступні 3 стандартні бібліотечні функції друку рядків: puts(), fputs() і printf().
 - Функції puts() достатньо передати в якості аргумента адресу рядка.

```
#include <stdio.h>
#define DEF "Я – строка, определенная директивой #define."
int main(void)
{
    char str1[80] = "Массив был инициализирован моим значением.";
    const char * str2 = "Указатель был инициализирован моим значением.";

    puts("Я – аргумент функции puts().");
    puts(DEF);
    puts(str1);
    puts(str2);
    puts(&str1[2]);
    puts(str2+4);

    return 0;
}
```

Я – аргумент функции puts().
Я – строка, определенная директивой #define.
Массив был инициализирован моим значением.
Указатель был инициализирован моим значением.
сив был инициализирован моим значением.
атель был инициализирован моим значением.

Функція puts()

- Припиняє ввід, коли стикається з нульовим символом, тому бажано, щоб у рядку він був присутнім.

```
/* Програма nono.c -- не делайте так! */  
#include <stdio.h>  
int main(void)  
{  
    char side_a[] = "Сторона А";  
    char dont[] = {'У', 'Р', 'А', '!' };  
    char side_b[] = "Сторона Б";  
  
    puts(dont);    /* dont не является строкой */  
    return 0;  
}
```

буде друкувати вміст пам'яті, яка межує з масивом dont, поки не знайде нульовий символ де-небудь в іншому місці.

Щоб нульовий символ не виявився надто далеко, масив dont поміщено між 2 справжніми рядками.

Вивід програми: ВАУ ! Сторона Б

Функція `fputs()`

- Приймає другий аргумент, що вказує на файл, у який слід виконувати запис.
 - Для виводу на пристрій відображення можна скористатись аргументом `stdout`, який визначається в заголовковому файлі `stdio.h`.
 - На відміну від `puts()`, функція `fputs()` не додає символ нового рядка у вихідні дані.
- Функція `gets()` відкидає символ нового рядка із вхідних даних, проте функція `puts()` додає цей символ у вихідні дані.
 - З іншого боку, функція `fgets()` зберігає цей символ, а `fputs()` не додає його у вихідні дані.

Функція fputs()

- Нехай потрібно побудувати цикл, який зчитує рядок та відтворює її в наступному рядку вихідних даних.
 - 1) gets() повертає нульовий вказівник (інтерпретується як false), як тільки зустрічає кінець файлу.

```
char line[81];  
while (gets(line))  
    puts(line);
```

- 2) можна виконати наступні оператори:

```
char line[81];  
while (fgets(line, 81, stdin))  
    fputs(line, stdout);
```

- Функція puts () розроблялась для роботи в парі з gets(), а функція fputs() – з fgets() .

Функція printf()

- Аналогічно до функції puts(), приймає адресу рядка в якості аргумента.
 - Менш зручна, ніж puts(), проте більш універсальна.
- Одна з відмінностей: printf() не друкує автоматично кожний рядок з нового рядка.
 - `printf ("%s\n", string);` призводить до того ж результату, що і `puts(string);`
- Функція printf() спрощує розміщення кількох рядків в одному рядку виводу.
 - `printf("Well, %s, %s\n" , name , MSG);`
 - MSG – символічний рядок, визначений за допомогою `#define`:

Рядкові функції (заголовковий файл string.h)

- Найбільш корисні:
 - `strlen()` – знаходить довжину рядка
 - `strcat()` – конкатенація (з'єднання) двох рядків
 - `strncat()` – конкатенація (з'єднання) двох рядків з перевіркою, чи вміщається другий рядок в перший масив
 - `strcmp()` – порівняння рядків до першого символу, який буде різним
 - `strncmp()` – порівняння рядків до першого символу, який буде різним, або за довжиною рядків
 - `strcpy()` – копіює рядок з тимчасового в постійне місце зберігання (рядковий еквівалент оператора присвоєння)
 - `strncpy()` – `strcpy()` з перевіркою, чи вміщається другий рядок в перший масив



test_fit.c

Функція strlen()

```
1 #include <stdio.h>
2 #include <string.h> /* contains string function prototypes */
3 void fit(char *, unsigned int);
4
5 int main(void)
6 {
7     char mesg[] = "Things should be as simple as possible,"
8     " but not simpler.";
9
10    puts(mesg);
11    fit(mesg, 38);
12    puts(mesg);
13    puts("Let's look at some more of the string.");
14    puts(mesg + 39);
15
16    return 0;
17 }
18
19 void fit(char *string, unsigned int size)
20 {
21     if (strlen(string) > size)
22         string[size] = '\0';
23 }
```

Things should be as simple as possible, but not simpler.
Things should be as simple as possible
Let's look at some more of the string.
but not simpler.

Исходная строка

П	р	е	о	д	о	л	е	в	а	й	т	е		т	р	у	д	н	о	с	т	и	.	\0
---	---	---	---	---	---	---	---	---	---	---	---	---	--	---	---	---	---	---	---	---	---	---	---	----

Строка после выполнения fit (mesg, 9)

П	р	е	о	д	о	л	е	в	\0	й	т	е		т	р	у	д	н	о	с	т	и	.	\0
---	---	---	---	---	---	---	---	---	----	---	---	---	--	---	---	---	---	---	---	---	---	---	---	----

Начало

Конец

Начало

Конец

puts (mesg) ;

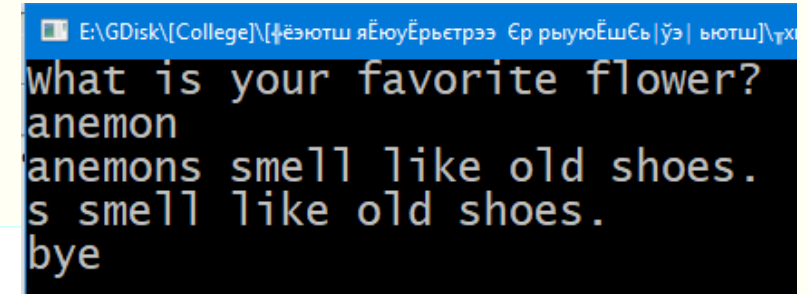
puts (mesg + 10) ;

Функція strcat()

- Копія другого рядка приєднується в кінець першого.
 - Другий рядок не змінюється.
 - Функція strcat() має тип char *, повертає значення свого першого аргумента — адресу першого символу рядка, в кінець якого було додано другий рядок.

```
1 #include <stdio.h>
2 #include <string.h> /* declares the strcat() function */
3 #define SIZE 80
4 char * s_gets(char * st, int n);
5 int main(void)
6 {
7     char flower[SIZE];
8     char addon[] = "s smell like old shoes.";
9
10    puts("What is your favorite flower?");
11    if (s_gets(flower, SIZE))
12    {
13        strcat(flower, addon);
14        puts(flower);
15        puts(addon);
16    }
17    else
18        puts("End of file encountered!");
19    puts("bye");
20
21
22    return 0;
23 }
```

```
26 char * s_gets(char * st, int n)
27 {
28     char * ret_val;
29     int i = 0;
30
31     ret_val = fgets(st, n, stdin);
32     if (ret_val)
33     {
34         while (st[i] != '\n' && st[i] != '\0')
35             i++;
36         if (st[i] == '\n')
37             st[i] = '\0';
38         else // must have words[i] == '\0'
39             while (getchar() != '\n')
40                 continue;
41     }
42     return ret_val;
43 }
```



```
E:\GDisK\[College]\{ёэюш яёюёрьстрээ ёр ыуюёшсь|ўэ| ьютш}\_x
what is your favorite flower?
anemon
anemons smell like old shoes.
s smell like old shoes.
bye
```

Функція strncat()

```
1 #include <stdio.h>
2 #include <string.h>
3 #define SIZE 30
4 #define BUGSIZE 13
5 char * s_gets(char * st, int n);
6 int main(void)
7 {
8     char flower[SIZE];
9     char addon[] = "s smell like old shoes.";
10    char bug[BUGSIZE];
11    int available;
12
13    puts("What is your favorite flower?");
14    s_gets(flower, SIZE);
15    if ((strlen(addon) + strlen(flower) + 1) <= SIZE)
16        strcat(flower, addon);
17    puts(flower);
18    puts("What is your favorite bug?");
19    s_gets(bug, BUGSIZE);
20    available = BUGSIZE - strlen(bug) - 1;
21    strncat(bug, addon, available);
22    puts(bug);
23
24    return 0;
25 }
```

- Приймає другий аргумент, який вказує максимальну кількість доданих символів.

Какой у вас любимый цветок?

Роза

Роза пахнет как старые валенки.

Какое у вас любимое насекомое?

Комар

Комар пахнет

- Інші функції



ДЯКУЮ ЗА УВАГУ!