

МІНІСТЕРСТВО НАУКИ ТА ОСВІТИ УКРАЇНИ
Національний Авіаційний Університет
Факультет комп'ютерних наук та технологій
Кафедра прикладної математики

“Алгоритмічні мови та програмування”
Лабораторна робота №4.4
29 травня 2023 р.

Лабораторну роботу виконав
студент №18 групи ПМ 1516 НАУ
Владислав Реган,
Викладач: Андрій Костянтинович Шевченко

Зміст

1	Постановка задачі	2
2	Теоретичні відомості	2
2.1	Умовний оператор if else	2
2.2	Цикл while	2
2.3	Цикл for	3
2.4	Функція printf()	3
	Параметри	3
2.5	Функція malloc()	4
	Поведінка функції	4
	Параметри	4
2.6	Функція calloc()	4
	Поведінка функції	4
	Параметри	4
2.7	Функція realloc()	4
	Параметри	5
2.8	Функція free()	5
	Поведінка функції	5
	Параметри	5
2.9	Функція strlen()	5
	Параметри	6
2.10	Фнкція strcat()	6
	Поведінка функції	6
	Параметри	6
2.11	Функція strcmp()	6
	Поведінка функції	6
	Параметри	6
2.12	Функція memchr()	6
	Поведінка функції	6
	Параметри	7
2.13	Функція memset()	7
	Поведінка функції	7
	Параметри	7
3	Тестування	8
4	Висновок	8
5	Теоретичні запитання	8
	Перелік ілюстрацій	9
	Перелік лістингів	9
	Література	10

1. Постановка задачі

Тема: Операція присвоювання і конструктор копіювання.

Мета: Переробити функцію сортування таблиці слів з лабораторної роботи 4.2 так, щоб вона використовувала системний шаблон `swap` (або власний шаблон для обміну значень). Розробити ще одну версію лабораторної роботи 4.2. Списки і таблиця слів. При цьому:

- ◇ Переробити структуру `wrd` в ускладнену версію (тобто так, щоб у ній був покажчик на символи, а не масив).
- ◇ Функцію сортування таблиці слів з лабораторної роботи 4.2 створити так, щоб вона використовувала системний шаблон `swap` (або власний шаблон для обміну значень).

Завдання

- ◇ Перевантажити операцію копіювання і конструктор присвоєння структури.
- ◇ У функції сортування таблиці слів використовувати системний шаблон `swap`.

2. Теоретичні відомості

2.1. Умове розгалуження `if else`

Якщо умова істинна, то відбувається перехід до тіла оператора `if`. В іншому випадку виконання програми переходить в оператор¹ після `else`. Оператор `if` виконує тільки один оператор в залежності від істинності умови: або оператор після `if`, або оператор після `else`.

```
if(condition) {  
    operator1;  
    operator2;  
    operator3;  
} else {  
    operator1;  
    operator2;  
    operator3;  
}
```

Ліс. 1: Синтаксис оператора умовного переходу `if else`

2.2. Оператор циклу `while`

Оператор циклу `while` виконується до того моменту, поки задані умови (вираз) залишаються істинними. На початку циклу обробляється умова і якщо вона виконується, то тоді вже виконується саме тіло циклу. Однак, на відміну від оператора `if`, після завершення тіла циклу, виконання повертається назад до `while` і перевірка умов повторюється. Якщо умова істинна - цикл повторюється. Якщо умова оператора `while` не виконується, то тоді тіло циклу пропускається і виконання програми переходить до першого оператора за циклом.

¹ або блок

```
while(condition) {  
    operator1;  
    operator2;  
    operator3;  
}
```

Ліс. 2: Синтаксис оператора циклу `while`

2.3. Оператор циклу `for`

Оператор циклу `for` зручно використовувати коли відома необхідна кількість дій. Тобто, в самому циклі спочатку перевіряється перша умова і якщо вона виконується - то тільки тоді перевіряється друга. В супротивному випадку виконання циклу зупиняється і здійснюється перехід до оператора, що знаходиться за тілом циклу. Потім виконується саме тіло циклу та після його виконання виконується `repeat`. Цикл буде продовжуватись до поки умова `condition` буде істинною.

```
for(one_time_action, condition, repeat) {  
    operator1;  
    operator2;  
    operator3;  
}
```

Ліс. 3: Синтаксис оператора циклу `for`

2.4. Функція `printf()`

Функція `printf()` використовується для виводу даних на дисплей. Вона отримує один або більше аргументів: перший аргумент – це дані які потрібно візуалізувати. Якщо потрібно вивести значення якоїсь змінної, то потрібно:

- Ⓐ зазначити вираз що починається зі знаку `%`
- Ⓑ вказати тип даних та формат для виведення

Самі змінні вказуються після зазначення типу. Також, якщо в кінці функції `printf()` зазначити символ `\n`, то в кінці виводу каретка перейде на новий рядок.

```
int printf(const char* format, ... );
```

Ліс. 4: Прототип фнккції `printf()`

Параметри

- ◇ `format` - вказівник на багатобайтовий рядок з закінченням `\0`, що визначає спосіб інтерпретації даних
- ◇ `...` - аргументи, що визначають дані для друку. Якщо будь-який аргумент після просування аргументу за замовчуванням не є типом, очікуваним відповідним специфікатором перетворення, або якщо аргументів менше, ніж вимагається форматом, то поведінка не визначена. Якщо аргументів більше, ніж вимагається форматом, зайві аргументи

обчислюються та ігноруються.

2.5. Функція `malloc()`

Функція `malloc()` виділяє пам'ять яка не використовуються. Якщо виділення пройшло успішно, то повертає вказівник, який можна привести до бажаного типу і використовувати виділену пам'ять через цей вказівник.

Поведінка функції

Якщо розмір `size` дорівнює нулю, поведінка `malloc()` визначається реалізацією. Наприклад, може бути повернутий нульовий вказівник `NULL`. Може бути повернутий ненульовий вказівник, але з таким вказівником працювати не варто, його треба передати в `free()`, щоб уникнути витоку пам'яті.

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

Ліс. 5: Прототип фнкції `malloc()`

Параметри

- ◇ `size` - кількість байт для виділення

2.6. Функція `calloc()`

`calloc()` виділяє пам'ять для масиву об'єктів розміром `num` та ініціалізує всі байти у виділеному сховищі нулем. Якщо виділення пройшло успішно, повертає вказівник на молодший (перший) байт у виділеному блоці пам'яті, який приведений належним чином для будь-якого типу.

Поведінка функції

Якщо `size` дорівнює нулю, то поведінка визначається реалізацією (може бути повернутий нульовий вказівник `NULL`, або деякий ненульовий вказівник, який не може бути використаний для доступу до пам'яті)

```
void *calloc(size_t num, size_t size);
```

Ліс. 6: Прототип фнкції `calloc()`

Параметри

- ◇ `num` - кількість об'єктів
- ◇ `size` - розмір кожного об'єкту

2.7. Функція `realloc()`

Фнкція `realloc()` перерозподіляє задану область пам'яті. Вона повинна бути попередньо виділена `malloc()`, `calloc()` або `realloc()` і ще не звільнена викликом `free()`. В іншому випадку результати будуть невизначеними. Перерозподіл здійснюється наступним чином:

- Ⓐ розширення або звуження існуючої області, на яку вказує `ptr`, якщо це можливо. Вміст області залишається незмінним до меншого з нових та старих розмірів. Якщо область

розширюється, то вміст нової частини масиву не визначається.

- Б) виділення нового блоку пам'яті розміром `new_size` байт, копіювання області пам'яті розміром, що дорівнює меншому з нового та старого розмірів, та звільнення старого блоку. Якщо пам'яті недостатньо, то старий блок пам'яті не звільняється і повертається нульовий вказівник `NULL`. Якщо `ptr` рівний `NULL`, то поведінка така сама як і при виклику `malloc(new_size)`.

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

Ліс. 7: Прототип фнкції `realloc()`

Параметри

- ◇ `ptr` - вказівник на ділянку пам'яті, що перерозподіляється
- ◇ `new_size` - новий розмір масиву в байтах

2.8. Функція `free()`

Функція `free()` звільняє місце, раніше виділене функціями `malloc()`, `calloc()`, або `realloc()`. Якщо `ptr` є нульовим покажчиком `NULL`, то функція нічого не робить.

Поведінка функції

Поведінка не визначена, якщо значення `ptr` не дорівнює значенню, повернутому раніше функціями `malloc()`, `calloc()`, `realloc()`. Також, поведінка не визначена, якщо область пам'яті, на яку посилається `ptr`, вже була звільнена, тобто функція `free()` або `realloc()` вже викликалась з аргументом `ptr` і жодні виклики `malloc()`, `calloc()`, `realloc()` не призвели до того, що після цього вказівник став рівним `ptr`. Поведінка не визначена, якщо після роботи `free()` до пам'яті здійснюється доступ через вказівник `ptr` (якщо тільки інша функція виділення не призвела до того, що значення вказівника стало рівним `ptr`).

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

Ліс. 8: Прототип фнкції `free()`

Параметри

- ◇ `ptr` - вказівник на пам'ять для звільнення

2.9. Функція `strlen()`

Функція `strlen()` повертає довжину заданого байтового рядка з нульовим закінченням, тобто кількість символів у символьному масиві, на перший елемент якого вказує `str`, до першого нульового символу не включно.

```
size_t strlen(const char *str);
```

Ліс. 9: Прототип фнкції `strlen()`

Параметри

- ◇ `str` - вказівник на рядок з нульовим символом `\0` в кінці, що підлягає перевірці

2.10. Функція `strcat()`

Функція `strcat()` копію рядка з нульовим закінченням, на який вказує `src`, в кінець байтового рядка з нульовим закінченням, на який вказує `dest`. Символ `src[0]` замінює нульовий термінатор в кінці `dest`. Результуючий байтовий закінчиться на `\0`.

Поведінка функції

Поведінка не визначена, якщо масив `dest` недостатньо великий для вмісту `src`, та немає місця для завершального нульового символу `\n`. Поведінка не визначена, якщо рядки перекриваються. Також, поведінка не визначена, якщо `dest` або `src` не є вказівником на рядок, що завершується на `\0`.

```
char *strcat(char *dest, const char *src);
```

Ліс. 10: Прототип функції `strcat()`

Параметри

- ◇ `dest` - вказівник на рядок з закінченням `\0`, до якого потрібно дописати рядок
- ◇ `src` - вказівник на байтовий рядок з закінченням `\0` для копіювання з нього

2.11. Функція `strcmp()`

Функція `strcmp()` порівнює два байтових рядки з нульовим закінченням лексикографічно. Знаком результату є знак різниці між значеннями першої пари символів (обидва інтерпретуються як `unsigned char`), які відрізняються у порівнюваних рядках.

Поведінка функції

Поведінка не визначена, якщо `lhs` або `rhs` не є вказівниками на рядки з нульовим закінченням `\0`.

```
int strcmp(const char *lhs, const char *rhs);
```

Ліс. 11: Прототип функції `strcmp()`

Параметри

- ◇ `lhs`, `rhs` - вказівники на рядки з закінченням `\0` для порівняння

2.12. Функція `memcpy()`

Поведінка функції

Поведінка не визначена, якщо доступ відбувається за межами кінця масиву `dest`. Якщо об'єкти перекриваються, то поведінка також не визначена. Поведінка не визначена, якщо або `dest`, або `src` є недопустимим або нульовим вказівником `NULL`.

```
void* memcpy(void *dest, const void *src, size_t count);
```

Ліс. 12: Прототип функції `memcpy()`

Параметри

- ◇ `dest` - вказівник на місце, куди треба зберегти данні
- ◇ `src` - вказівник на місце, звідки брати дані для копіювання
- ◇ `count` - кількість байтів, які потрібно скопіювати

Функція `memcpy()` копіює `count` символів з об'єкту на який вказує `src` у `dest`. Обидва масиви інтерпретуються як масиви `unsigned char`.

2.13. Функція `memset()`

Функція `memset()` копіює значення `(unsigned char)ch` в кожен з перших `count` символів об'єкту, на який вказує `dest`.

Поведінка функції

Поведінка не визначена, якщо доступ відбувається за межами кінця масиву `dest`. Також, поведінка не визначена, якщо `dest` є нульовим вказівником `NULL`.

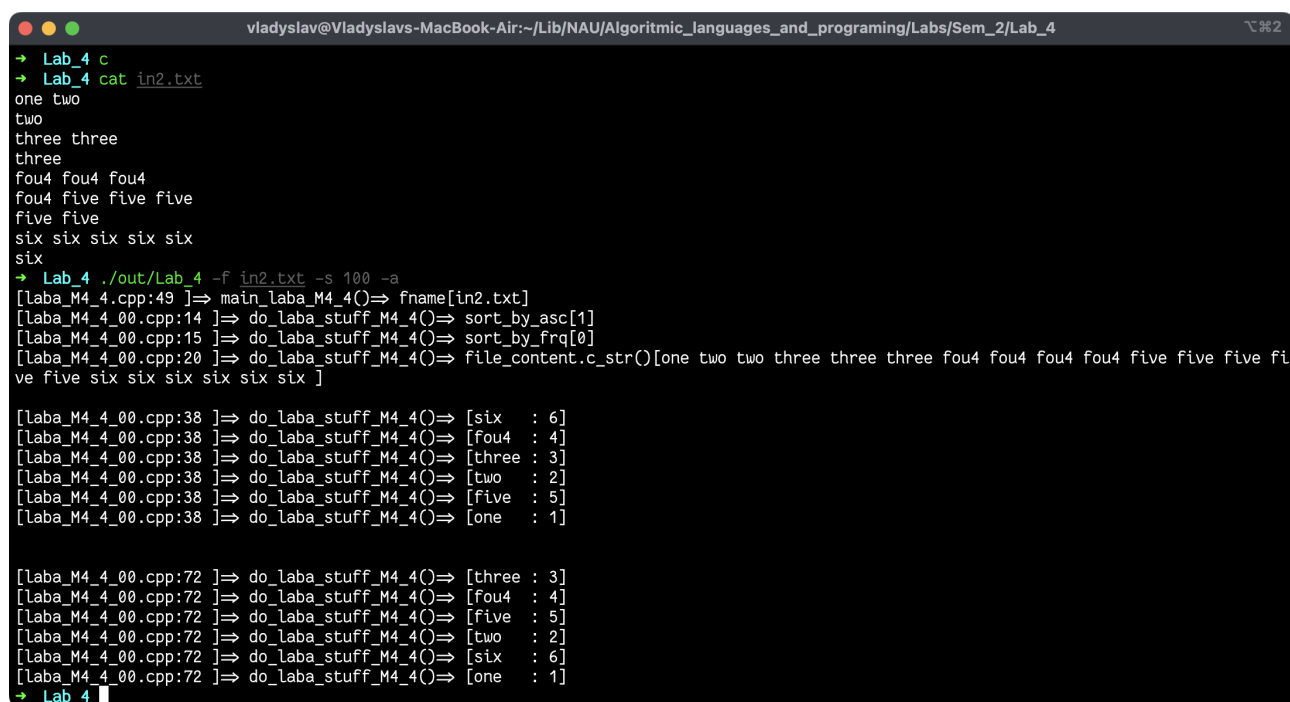
```
void *memset(void *dest, int ch, size_t count);
```

Ліс. 13: Прототип функції `memset()`

Параметри

- ◇ `dest` - вказівник на об'єкт для заповнення
- ◇ `ch` - байт для заповнення
- ◇ `count` - кількість байтів, які потрібно заповнити

3. Тестування



```
vladyslav@Vladyslavs-MacBook-Air:~/Lib/NAU/Algorithmic_languages_and_programing/Labs/Sem_2/Lab_4
→ Lab_4 c
→ Lab_4 cat in2.txt
one two
two
three three
three
fou4 fou4 fou4
fou4 five five five
five five
six six six six six
six
→ Lab_4 ./out/Lab_4 -f in2.txt -s 100 -a
[laba_M4_4.cpp:49 ]⇒ main_laba_M4_4()⇒ fname[in2.txt]
[laba_M4_4_00.cpp:14 ]⇒ do_laba_stuff_M4_4()⇒ sort_by_asc[1]
[laba_M4_4_00.cpp:15 ]⇒ do_laba_stuff_M4_4()⇒ sort_by_frq[0]
[laba_M4_4_00.cpp:20 ]⇒ do_laba_stuff_M4_4()⇒ file_content.c_str()[one two two three three three fou4 fou4 fou4 fou4 five five five fi
ve five six six six six six six ]

[laba_M4_4_00.cpp:38 ]⇒ do_laba_stuff_M4_4()⇒ [six : 6]
[laba_M4_4_00.cpp:38 ]⇒ do_laba_stuff_M4_4()⇒ [fou4 : 4]
[laba_M4_4_00.cpp:38 ]⇒ do_laba_stuff_M4_4()⇒ [three : 3]
[laba_M4_4_00.cpp:38 ]⇒ do_laba_stuff_M4_4()⇒ [two : 2]
[laba_M4_4_00.cpp:38 ]⇒ do_laba_stuff_M4_4()⇒ [five : 5]
[laba_M4_4_00.cpp:38 ]⇒ do_laba_stuff_M4_4()⇒ [one : 1]

[laba_M4_4_00.cpp:72 ]⇒ do_laba_stuff_M4_4()⇒ [three : 3]
[laba_M4_4_00.cpp:72 ]⇒ do_laba_stuff_M4_4()⇒ [fou4 : 4]
[laba_M4_4_00.cpp:72 ]⇒ do_laba_stuff_M4_4()⇒ [five : 5]
[laba_M4_4_00.cpp:72 ]⇒ do_laba_stuff_M4_4()⇒ [two : 2]
[laba_M4_4_00.cpp:72 ]⇒ do_laba_stuff_M4_4()⇒ [six : 6]
[laba_M4_4_00.cpp:72 ]⇒ do_laba_stuff_M4_4()⇒ [one : 1]
→ Lab_4
```

Рис. 1: Тестування

4. Висновок

Програма працює як і заплановано.

5. Теоретичні запитання

Чим текстовий файл відрізняється від двійкового файлу?

Двійкові файли зберігають інформацію у вигляді інформації в пам'яті комп'ютера під час роботи програми, що дозволяє не виконувати жодних перетворень, прискорюючи цим процес читання. Також бінарні формати мають маркетингову перевагу, оскільки відсутність чітко описаної специфікації створення дозволяє утруднити його розуміння. Крім переваги, двійкові файли мають величезний недолік, пов'язаний із їх сумісністю. Тому двійковий файл може по-різному прочитатися на різних операційних системах. Текстові файли є універсальним засобом представлення інформації у зв'язку з їх сумісністю. Тобто, незалежно від різних операційних систем, інформація буде прочитана правильно, крім проблем із кодуванням. Також перевагою текстових файлів є незалежність від порядку байт у слові. Саме тому всі стандарти чи протоколи передачі в Internet є текстовими.

Перелік ілюстрацій

1	Тестування	8
---	----------------------	---

Перелік лістингів

Постановка задачі	2
-------------------	---

Теоретичні відомості	2
----------------------	---

1	Синтаксис if else	2
2	Синтаксис while	3
3	Синтаксис for	3
4	Прототип printf()	3
5	Прототип malloc()	4
6	Прототип calloc()	4
7	Прототип realloc()	5
8	Прототип free()	5
9	Прототип strlen()	5
10	Прототип strcat()	6
11	Прототип strcmp()	6
12	Прототип memcpy()	7
13	Прототип memset()	7

Література

- [1] О. V. Gavrulenko. О. G. Piskunov О. Р. Tomachuk. *Алгоритмічні мови та програмування: лабораторний практикум*. Т. 40. Україна, м.Київ: НАУ, 2022.
- [2] О. G. Piskunov. *Конспект лекцій: Компілятори C++ та розробка консольних додатків*. 2. Україна, м.Київ: НАУ, груд. 2020.
- [3] Stephen Prata. *C Primer Plus, 6th edition*. 2016.
- [4] Richard Stallman та ін. *GNU C library reference manual version 2.26*. 12TH MEDIA SERVICES, 2018.