

Тема 05. Вступ до розробці з використанням динамічного керування пам'ятті

Давидов В.В.

Сегменти пам'яті

- Сегмент коду
- Сегмент даних
 - Статичні та глобальні змінні, на які можна отримати прямий доступ на протязі всього життєвого циклу програми. Їх розмір відомий (і постійний) при запуску програми
- Сегмент стеку
 - Локальні змінні функцій, параметри функції, "стек виклику функцій", масиви змінної довжини (VLA)
- Сегмент кучі (heap)
 - Динамічно створені дані
 - Макс 2Гб (Си)

Виділення та звільнення пам'яті (мова C)

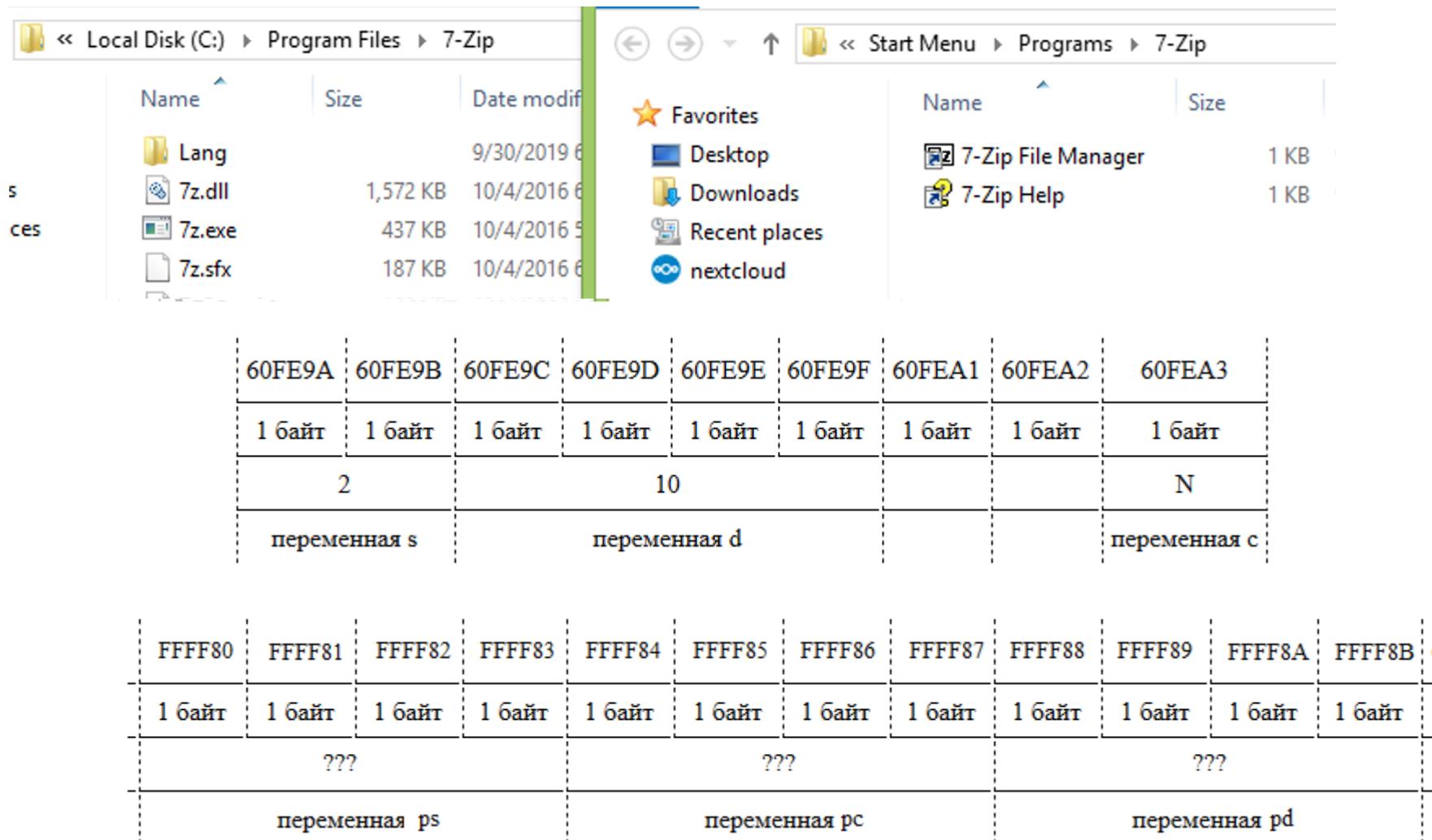
- `#include <stdlib.h>`
 - `void *malloc(size_t size)`
 - `int * p = (int *) malloc(x * sizeof(int));`
 - `void *calloc(size_t num, size_t size)`
 - `int * p = (int *) calloc(x, sizeof(int));`
 - `void *realloc(void *ptr, size_t newsize)`
 - `int * v = (int *) realloc(p, y * sizeof(int));`
 - Небажано його використовувати через неоднозначність розміщення результуючої пам'яті (розширення існуючої або виділення в новому місці)
 - `void free(void *ptr)`
 - `free (p);`
- `#include <malloc.h>`
 - `void *alloca(size_t size)`

Виділення та звільнення пам'яті (мова С). Інше

- Завжди перевіряйте, що повертають функції виділення пам'яті (чи результат дорівнює 0)
- `void*` - вказівник на типизовану область пам'яті. «Сира пам'ять»
- `free` – не очищає фізично дані, лише вказує менеджеру пам'яті, що це блок пам'яті можна використовувати іншими процесами. Для безпечної роботи (сенситивних даних), після звільнення пам'яті, вкрай бажано занулювати значення вказівника, щоб уникнути доступу до звільненої пам'яті
- (занадто перебільшено) Кількість викликів функції виділення пам'яті повинна бути ідентичною кількістю викликів функції звільнення пам'яті

Робота з показчиками

```
short s = 2;
int d = 10;
char c = 'N';
// ...
short *ps = &s;
// short *ps; ps = &s;
char *pc = &c;
int *pd = &d;
// ...
*pd = 45;
// ...
pc = 0x60FE9C;
*pc = 25; // ???
(Працюватиме не
як очікується)
```



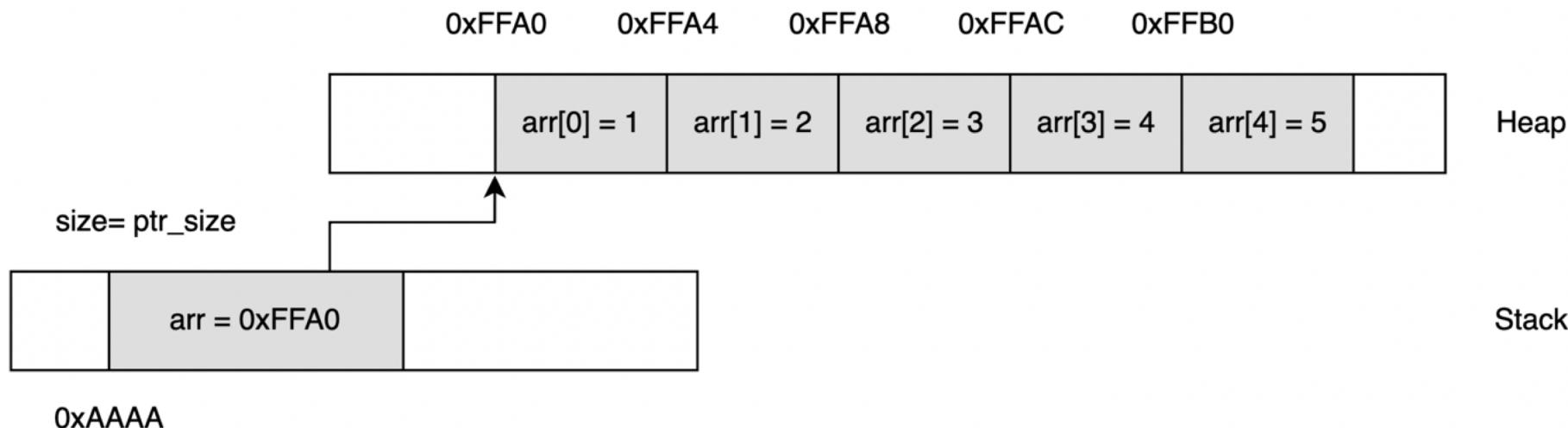
Робота з динамічними об'єктами

- Робота з динамічними масивами

```
int size = 5; // not constant
```

```
int *arr = (int*) malloc(size * sizeof(int));
```

```
for (int i = 0; i < size; i++) {  
    *(arr + i) = 0; // arr[i] = 0; // + i ==> + i * sizeof(int)  
}
```



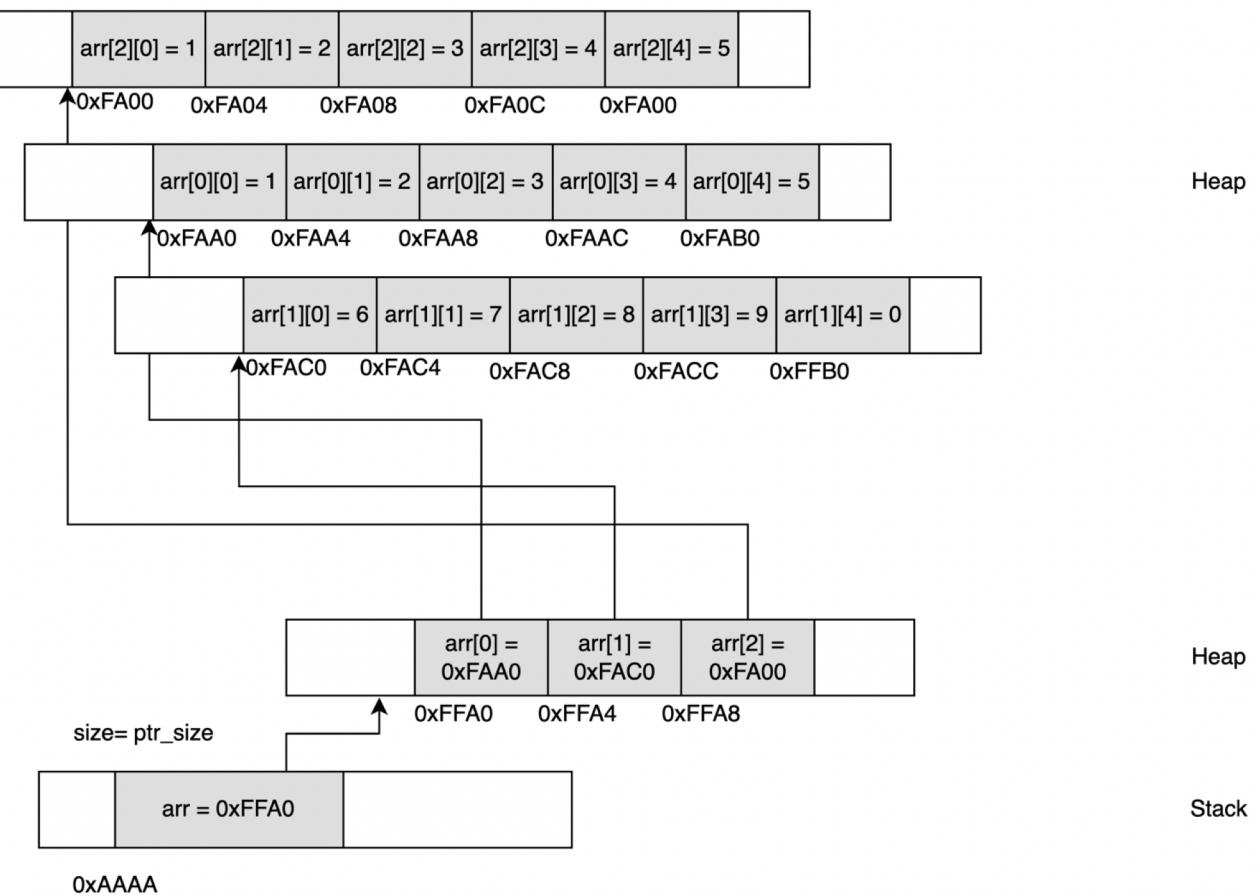
Робота з динамічними об'єктами

- Вказівники на функції
 - int get_min(int a, int b);
 - int (*func_ptr)(int, int);
 - func_ptr = &get_max; // func_ptr = get_max;
 - int x = (*func_ptr)(4, 5); // int x = func_ptr(4, 5);

Робота з динамічними об'єктами

- Робота з динамічними масивами (частина 2): (масив покажчиків на масив покажчиків)

```
int sizeX = 5; // not constant
int sizeY = 10;
int **arr = (int**) malloc(sizeX * sizeof(int*));
// sizeX * size (ptr)
for (int i = 0; i < sizeX; i++) {
    *(arr + i) = (int*) malloc(sizeY * sizeof(int));
}
for (int i = 0; i < sizeX; i++) {
    for (int j = 0; j < sizeY; j++) {
        *(*(arr + i) + j) = 0; // arr[i][j] = 0;
    }
}
for (int i = 0; i < sizeX; i++) {
    free(*(arr + i));
}
free(arr);
```



- Перспектива: Розріджені масиви

Виведення динамічних об'єктів у відладчику

- `int **arr = (int**) malloc(3 * sizeof(int*));`
 - `*(arr + i) = (int*) malloc(4 * sizeof(int));`
- `int *mas = (int*) malloc(3 * sizeof(int));`
- `int x = 3; int *var = &x;`

Expression:
`(*int(*)[3]) mas`

Result:
└ result = {int [3]}
 └ [0] = {int} 0
 └ [1] = {int} 1
 └ [2] = {int} 2

Expression:
`var`

Result:
└ result = {int *} 0x3052b5860
 └ *\$15 = {int} 3

```
(lldb) p array 3 mas
(int *) $0 = 0x0000000100206a50 {
    (int) [0] = 0
    (int) [1] = 1
    (int) [2] = 2
}
(lldb) p *(int(*)[3]) arr[0]
(int [3]) $1 = ([0] = 0, [1] = 1, [2] = 2)
(lldb) p var
(int *) $2 = 0x000000016fdff7b0
(lldb) p *var
(int) $3 = 3
```

Особливості: виведення багатовимірного динамічного масиву (масиву масивів) неможливе. Необхідно виводити його набором одновимірних масивів

Передача даних у функції «за вказівником»

- void noChange(int a) { a = 300; }
- void changeMe(int *a) { *a = 300; }
- void allocateArrayWrong(int *a) { a = (int*) malloc(100500); }
- void allocateArrayGood(int **a) { *a = (int*) malloc(100500); }
- int main() {
 int a = 50; int * b;
 changeMe(&a); // wrong: noChange(a);
 allocateGood(&b); // wrong: allocateWrong(b);
 free(b);
}
- **NOTE:**
 - Усі масиви передаються «автоматично» за вказівником
 - «N-мірні» динамічні масиви передаються з N-им кілом «зірочок». А якщо ми хочемо виділяти/звільнити для нього пам'ять – то передаються з N+1 кількома «зірочками»
 - Хороша практика – звільнити пам'ять у тій області видимості, як і створення, тому виконувати функції лише для виділення пам'яті під масив – погана ідея

Думки щодо вказівників, які спростяють вам життя

- Вказівники - просто числа
 - До них застосовні будь-які математичні операції, але у 99% випадків використовуються тільки операції + та -
- Тип покажчика визначає тип даних, який він розіменовується і якого діє адресна арифметика
 - `char * str = "Hello";`
 - `// *(str + 1) = 'ello'`,
 - `// *((int*) str) = 0x 6C 6C 65 48 = 'I' 'I' 'e' 'H'`
 - `// *((int*) str + 1) = 0x111 = 'o'`
- Повернення покажчиків із функції категорично небажане, бо важко визначити межі життя даного об'єкта
 - По сигнатурі функції не зрозуміло – ми повертаємо покажчик на один елемент чи масив
 - Якщо ми повертаємо масив – незрозумілий його розмір
 - Проблематично узгоджувати «хто звільнитиме пам'ять»

ВИТОКИ ПАМ'ЯТІ

- Valgrind
 - Compile with Debug info
 - valgrind --tool=memcheck --leak-check=full dist/main.bin
- llvm (13+) leak sanitizer (the only thing supported for M1 (jan-2022))
 - clang -fsanitize=address -g src/main.c -o dist/main.bin
 - ASAN_OPTIONS=detect_leaks=1 dist/main.bin
- _CrtDumpMemoryLeak (MSVC Only)
 - #define _CRTDBG_MAP_ALLOC
 - #include <stdlib.h>
 - #include <crtDBG.h>
 - ...
 - _CrtDumpMemoryLeaks();
- <https://drmemory.org/>

Typedef keyword

- **typedef** is used to define new type. It doesn't create new type but only make alias (synonym) for existing ones

- **typedef <existing_type> <new_type>**
 - **typedef unsigned int size_t**

- Similar to define:

```
#define FLOAT_PTR float *
FLOAT_PTR PF1, PF2; // float * PF1, PF2;
```

```
typedef float* float_ptr
float_ptr pf1, pf2; /* float *pf1, *pf2; */
```

- Distinguish:

- Visibility level: **typedef** has similar visibility level like ordinal variable has (according to curve brackets) ; **#define** is visible until **#undef** comes

Typedef keyword

- Useful examples:
 - `typedef unsigned int size_t`
- Don't invent wheels:
 - `#include <stddef.h>`
 - `typedef unsigned short char16_t;`
 - `typedef unsigned int char32_t;`
 - `typedef unsigned int size_t;`

Typedef and functions

- `typedef void (*myfunc)(); /* myfunc is a function that returns void and get no arguments */`
- `myfunc f = &func; /* compile equally as void (*f)(); */`
- `f(); /* call created function */`
- `typedef int (*t_somefunc)(int,int); /* t_somefunc is a function that returns int and get 2 int arguments */`
- `int product(int u, int v) { return u*v; }`
- `t_somefunc afunc = &product;`
- `int x2 = (*afunc)(123, 456); /* call product(int,int) to calculate 123*456 */`

Введення/виведення даних

- `#include <stdio.h>`
- Файл – іменована область даних на диску
- Консоль – теж файл. З заздалегідь визначені об'єкти файлу для роботи з консоллю/клавіатурою : `stdin`, `stdout`, `stderr`
- Неформатоване введення/виведення (низькорівневе):
 - `size_t fread(void *buffer, size_t oneItemSize, size_t itemCount, FILE *fp);`
 - `char str[6]; fread(&str, sizeof(char), 5, stdin); // Hello -> Hello`
 - `int a; fread(&a, 1, 1, stdin); // 0 -> 48`
 - `int a; fread(&a, sizeof(int), 1, stdin); // 1234 -> 875770417;`
 - `1234 -> 49 50 51 52 -> (hex) 31 32 33 34 -> 34 33 32 31 -> 875770417`
 - `size_t fwrite(const void * buffer, size_t oneItemSize, size_t itemCount, FILE *fp);`
 - `char str[] = "Hello"; fwrite(&str, sizeof(char), strlen(str), stderr); // Hello`
 - `int a = 110; fwrite(&a, sizeof(a), 1, stderr); // n`

Введення/виведення даних

- Неформатоване введення/виведення (високорівневе):
 - int fgetc(FILE *pointer)
 - char a1 = fgetc(stdin); // 0 -> '0'
 - fflush(stdin);
 - int a2 = fgetc(stdin); // 0 -> 48
 - int fputc(int char, FILE *pointer)
 - fputc('X', stderr); // -> X
 - fputc(52, stderr); // -> 4
 - int fputs(const char *str, FILE *fp);
 - fputs("Hello", stderr);
 - char *fgets(char *str, int len, FILE *fp);
 - char str[5]; fgets(str, 5, stdin); // Hello -> Hell
 - Аналоги вищезгаданих функцій без префікса «f» у імені функції та без другого аргументу

ASCII Table

- ASCII — American Standard Code for Information Interchange.
- ASCII була розроблена (1963) для кодування символів, коди яких поміщалися в 7 біт (128 символів). Згодом кодування було розширене до 8 біт (256 символів), коди перших 128 символів не змінилися.
- Керуючі символи ASCII (код символу 0-31)
- Друковані символи ASCII (код символу 32-127)
- Розширені символи ASCII (код символу 128-255) (e.g., Win-1251 - Cyrillic)
- Розвиток – UTF-8 кодування (1 символ займає від 1 до 4 байт, перші 127 біт - ASCII)

ASCII Table

ASCII Code Chart

	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	!	"	#	\$	%	&	'	()	+	+	,	-	'	/	
3	0	1	2	3	4	5	6	7	8	9	:	:	<	=	>	?
4	@	À	฿	₵	₵	₵	₵	₵	₵	₵	₵	₵	₵	₵	₵	₵
5	P	Q	R	S	T	U	₩	₩	₩	₩	Z	[\]	~	-
6	~	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	ର	ସ	ତ	୭	୮	୯	ୱ	୳	୳	{		}	~	DEL

У циклі виведіть значення "а" від 32 до 255 за допомогою
fwrite(&a, sizeof(a), 1, stderr);
І дізнайтесь, що у вас є після номера 127 ;)

[Форматоване] Введення/виведення даних

- int printf(const char *format, ...)
 - printf ("Hello, World");
- int scanf(const char *format, ...)
- printf/scanf family:
 - int fscanf (FILE * stream, const char * format, ...);
 - int sscanf (const char * s, const char * format, ...);
 - int fprintf (FILE * stream, const char * format, ...);
 - int snprintf (char * s, size_t n, const char * format, ...);
 - int sprintf (char * str, const char * format, ...);

[Форматоване] Введення/виведення даних

- Рядок формату – шаблон, який має «місця для спеціальної вставки». Складається з :
 - Специфікатори формату («місця для спеціальної вставки»)
 - Спеціальні символи (напр., \, /, ‘, “, \t, \n)
 - Інші символи (Основне тіло шаблону)
- Пример:
 - Student [x] from group [x] received [x] mark for Programing

[Форматоване] Введення/виведення даних

- Специфікатор Формата
 - %[флаги][ширина][.точність]специфікатор
 - Флаги:
 - - вирівнювання по лівому краю для заданої ширини тексту
 - + (Для чисел) примусово виставляє знак (в т.ч. позитивного)
 - # (для чисел) додавання префікса системи обчислень
 - 0 (для чисел) для заданої ширини тексту заповнювач вільного простору – нулі
 - Ширина – мінімальна кількість символів, які будуть виводитися
 - Точність – (для речових чисел). Кількість чисел після десяткового роздільника (за замовчуванням – 6)

[Форматоване] Введення/виведення даних

- Специфікатор Формата
 - %[флаги][ширина].[точність]специфікатор
 - Специфікатор
 - d, i – decimal value
 - u – unsigned decimal
 - o – octal value
 - X, x – hex value (uppercase, lowercase)
 - F, f, - float value
 - E, e – scientific format (3.14e+0)
 - 100500 -> 1.005 * 10⁵ -> 1.005e+5
 - 0.0125 -> 1.25 * 10⁻² -> 1.25e-2
 - c – character
 - s – string (character array)
 - p – pointer address

[Форматоване] Введення/виведення даних

- Примеры printf:

- `printf("Student %s from group %s received %d mark for Programing\n", "Sidorov", "CIT-999a", 99);`
 - Student Sidorov from group CIT-999a received 99 mark for Programing
- `printf("%10s | %-10s", "Hello", "Hello");`
 -Hello | Hello.....
- `printf("%c %d %+d %o %x %X %#x %u", ';', 10, 10, 10, 10, 10, 10, -10);`
 - ; 10 +10 12 a A 0xa 4294967286
- `printf("Pointer of main function: %p", &main);`
 - Pointer of main function: 0x10a78bc50
- `printf("%f %.2f %010.2f %e %.2e", 3.14159f, 3.14159f, 3.14159f, 3.14159f, 3.14159f);`
 - 3.141590 3.14 0000003.14 3.141590e+00 3.14e+00

[Форматоване] Введення/виведення даних

- Примеры scanf:
 - int a; char b2[10]; int array[6]; int *a2;
 - scanf("%d", &a); // 123 -> 123
 - scanf("%s", b2); // Hello -> Hello
 - scanf("%x", &a); // 100 -> ???
 - scanf("Hello %d", &a); // 123 -> 0 ; Hello 123 -> 123
 - scanf("%d %s", &a, b2);
 - scanf("%d", &array[i]);
 - scanf("%d", (a2 + i));
- NOTE: (in MSVC) scanf is deprecated (because of strings). Use scanf_s instead
- <https://man7.org/linux/man-pages/man3/printf.3.html>
- <https://man7.org/linux/man-pages/man3/scanf.3.html>

Робота з файлами

- Дії під час роботи з файлами:
 - Спробувати відкрити файл для читання або запису
 - FILE *file = fopen("path", "mode");
 - Mode:
 - r – read only (file existence required)
 - w – write only
 - a – append
 - b (as suffix: rb, wb, ab) – binary (how it is stored in memory, see fwrite)
 - (fopen_s is better, but it is not POSIX (MSVC only))
 - Не можна використовувати тільку (~) у шляхах
 - Переконайтесь, що ми отримали доступ до файлу
 - if (file != NULL) ...
 - Виконати операцію з файлом (читання, запис)
 - fprintf(file, "Hello, world");
 - while (!feof(file)) doRead;
 - Закрити файл (зі зберігання змін)
 - fclose(file);
- Результат операції роботи з файлом зберігається в змінній errno (оголошено в **errno.h**). Її текстове значення можна отримати за допомогою функції `char* strerror(int errnum);` (оголошено в **string.h**)

Приклади роботи з файлами

- Визначення розміру файлу (без перевірок)

- Варіант 1

- FILE *file = fopen("/Users/davs/test.lab", "r");
 - fseek(file, 0, SEEK_END);
 - long count = ftell(file);
 - printf("Filesize: %ld \n", count);
 - fclose(file);

- Варіант 2

- FILE *file = fopen("/Users/davs/test.lab", "r");
 - long count = 0;
 - while (!feof(file)) {
 - fgetc(file);
 - count++;
 - }
 - count--; // feof - is also hidden character
 - printf("Filesize: %ld \n", count);
 - fclose(file);

Linux streaming to input data

Code:

```
int n;  
scanf("%d", &n);  
int ** arr = malloc(n * sizeof(int*));  
for (int i = 0; i < n; i++)  
    arr[i] = malloc(n * sizeof(int));  
for (int i = 0; i < n; i++)  
    for (int j = 0; j < n; j++) scanf("%d", &arr[i][j]);  
  
for (int i = 0; i < n; i++) {  
    for (int j = 0; j < n; j++) printf("%d ", arr[i][j]);  
    printf("\n");  
}
```

Program execution:

3
1
2
3
4
5
6
7
8
9
1 2 3
4 5 6
7 8 9

Input.txt:

3
2 4 6
1 3 5
0 9 8

How to use:

1. cat input.txt | ./main.bin
2. ./main.bin < input.txt

Null Terminated C Strings

- NTSC = масив символів, останній символ = 0 (\0)
 - Навіщо? Щоб визначати межі даного масиву та не передавати у всі функції ці розміри
 - `char string[] = "Hello"; // --> char string[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };`
 - `char * string = "Hello"; // --> const char string[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };`
- Перетворення рядка на число і навпаки:
 - `int value2 = strtol("123456", &str2, 10); // "123456" -> 123456`
 - `sprintf(str, "%d", value); // int value=123; -> char str [5] => "123\0"`

Функції роботи з NTSC

#include <ctype.h> : Функції роботи з 1 символом (всі функції приймають `int c` та повертають int (прапор стану, або дані)

isalnum	Якщо аргумент функції є чи літерою, чи цифрою, вона повертає ненульове значення.
isalpha	Повертає ненульове значення, якщо її аргумент є буквою, інакше повертається нуль.
isdigit	Повертає ненульове значення, якщо її аргумент є десятковою цифрою, інакше повертається нуль.
isgraph	Повертає true, якщо це - друкований символ, який відрізняється від пробілу.
islower	Повертає true, якщо - символ нижнього регістра
isprint	Повертає true, якщо - символ, що друкується
ispunct	Повертає true, якщо - розділовий знак (будь-який друкований символ, відмінний від пробілу або алфавітно-цифрового символу).
isspace	Повертає true, якщо - пробіловий символ: пробіл, \n, \r, \t, \v
isupper	Повертає true, якщо - символ верхнього регістра.
isxdigit	Повертає true, якщо - шістнадцяткова цифра
toupper	переводить літеру нижнього регістру у верхній регістр
tolower	переводить літеру верхнього регістру до нижнього регістру

Функції роботи з NTSC

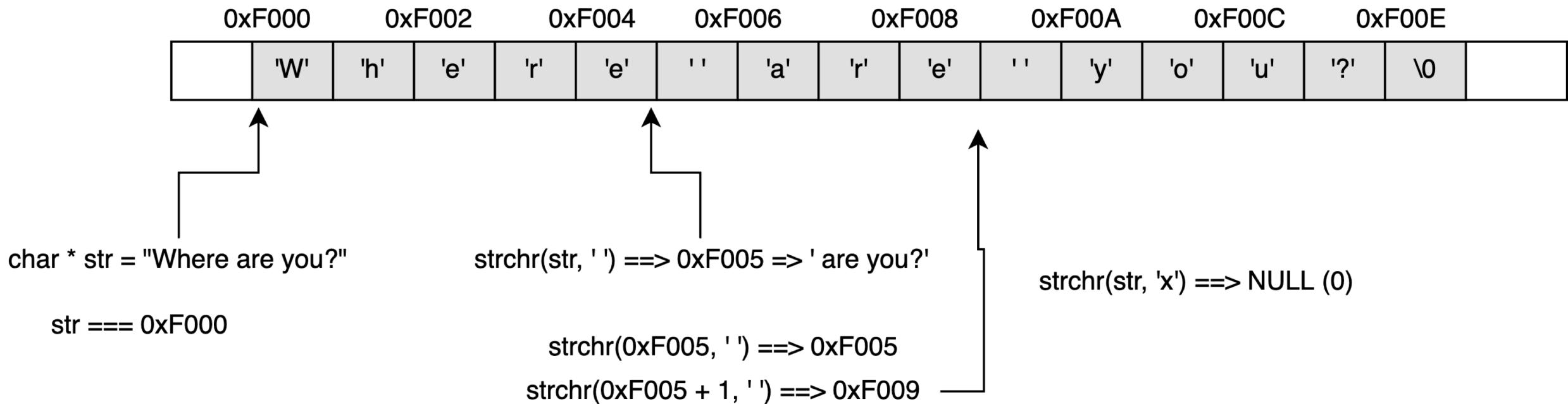
#include <string.h>

#include <string.h>	
char *strcat(char *dest, const char *src);	дописує рядок src у кінець dest
char *strchr(const char *s, int c);	повертає адресу символу с у рядку s, починаючи з голови, або NULL, якщо рядок s не містить символу с
char * strrchr(const char *s, int c);	повертає адресу символу с у рядку s, починаючи з хвоста, або NULL, якщо рядок s не містить символу с
int strcmp(const char *, const char *);	порівняння рядків (повертає "0", якщо рядки одинакові)
char *strcpy(char *toHere, const char *fromHere);	копіює рядок з одного місця до іншого
size_t strlen(const char *);	повертає довжину рядка (без урахування термінального символу)
char *strstr(const char *h, const char *needle);	знаходить перше входження рядка needle у h
char *strupr(const char *);	створює копію рядка та повертає покажчик на неї

Other:

- `strcasecmp(char*, char*)`, `strcasestr(char*, char*)` – process correspond functions ignoring case
- `strncat(char*, char*, size_t n)`, `strncmp(char*, char*, size_t n)`, `strncpy(char*, char*, size_t n)` – process correspond functions for first n characters

How strchr works



Модифікація рядків

- Видалення частини рядка. Наприклад, довжина рядка = 30, видалити символи з 10 до 20 (10 шт.)
 - Створити новий рядок із `strlen(original) - 10 + 1` символів К
 - зкопіювати через strncpy перші 9 символів з original у цільовий рядок
 - Додати в кінець цільового рядка (через strncat) символи 20+ оригінального рядка
 - Звільнити пам'ять оригінального рядка
 - Переприсвоєти покажчик цільового рядка на оригінальний

Модифікація рядків

- Вставка у середину рядка. Наприклад, довжина рядка = 30, вставити в позицію 10, рядок із 20 символів
 - Створити новий рядок із `strlen(original) + 20 + 1` символів
 - Зкопіювати через strncpy перші 9 символів з original у цільовий рядок
 - Додати до кінця цільового рядка (через strncat) (20 символів) додаткового рядка
 - Додати в кінець цільового рядка (через strncat) символи 10+ оригінального рядка
 - Звільнити пам'ять оригінального рядка
 - Переприсвоєти покажчик цільового рядка на оригінальний

Bonus. Replace substring in string

```
void replace(char * o_string, char * s_string, char * r_string) {  
  
    char buffer[MAX_L];                                // a buffer variable to do all replace things  
    char * ch;                                         // to store the pointer returned from strstr  
  
    if(!(ch = strstr(o_string, s_string)))             //first exit condition  
        return;  
  
    strncpy(buffer, o_string, ch-o_string);           //copy all the content to buffer before the first occurrence of the search string  
    buffer[ch-o_string] = 0;                            //prepare the buffer for appending by adding a null to the end of it  
  
    sprintf(buffer+(ch - o_string), "%s%s", r_string, ch + strlen(s_string));      //append using sprintf function  
  
    o_string[0] = 0;                                    // empty o_string for copying  
    strcpy(o_string, buffer);  
  
    return replace(o_string, s_string, r_string);       //pass recursively to replace other occurrences  
}
```

Bonus. Get count of character in string via strchr

```
#include <stdlib.h>
#include <string.h>

size_t get_count_symbol(const char* string, char symbol) {
    size_t count_symbol = 0;
    while ((string = strchr(string, symbol)) != NULL) {
        count_symbol++;
        string++;
    }
    return count_symbol;
}

int main() {
#define SYMBOL 'a'
#define ORIGINAL "abrákadábra"
#define MAX_LEN 100

    char *string = (char *) malloc(MAX_LEN * sizeof(char));
    strncpy(string, ORIGINAL, MAX_LEN);
    size_t cnt = get_count_symbol(string, SYMBOL);
    free(string);

    return 0;
}
```

```
START_TEST(test_get_count_symbol)
{
    #define DATA_SIZE 3

    char *input_data_str[DATA_SIZE] = { "abrákadábra",
"test_me", "abcd"};
    char input_data_sym[DATA_SIZE] = { 'a', 's', 'e'};
    size_t expected_values[DATA_SIZE] = { 5, 1, 0};

    for (int i = 0; i < DATA_SIZE; i++) {
        size_t actual_value =
get_count_symbol(input_data_str[i], input_data_sym[i]);
        ck_assert_int_eq(expected_values[i], actual_value);
    }
}
END_TEST
```

Strings. Hints

- Getline sample

```
char *line = NULL;  
size_t n = 0;  
ssize_t result = getline(&line, &n, stdin);  
printf("result = %zd, n = %zu, line = \"%s\"\n", result, n, line);  
free(line);
```

- itoa alternative – gcvt, sprintf (recommended)
- strtol: No Need to free buffer, as it is being pointed to existed string only
- Don't use strncpy (unless you really know what you're doing) --> use strdup

Робота із структурованими типами даних

- Структура – об'єднання різних змінних із, можливо, різними типами, які характеризують певний об'єкт.
- Приклад: об'єкт «співробітник», який характеризується ПІБ, посадою, зарплатою. З точки зору мови програмування, кожна характеристика – поле об'єкта.

```
struct Employee {  
    char fio[50];  
    double salary;  
    char position[20];  
}; /* <-- note: semicolon is required*/
```

Linus Torvalds about structs and typedef:
<https://yarchive.net/comp/linux/typedefs.html>

Structures initialization and fields access

```
/*create instance of Employer with individual  
characteristics */  
  
int main()  
{  
    struct Employee ivanov;  
    ivanov.salary = 100500;  
    strcpy(ivanov.fio, "Ivanov Ivan Ivanovich");  
    strcpy(ivanov.position, "Developer");  
  
    struct Employee petrov = {"Petrov Petr", 2000, "QA" };  
  
    printEmployee(&ivanov);  
    printEmployee(&petrov);  
  
    double ivanovSalary = ivanov.salary;  
    return 0;  
}
```

```
void printEmployee(struct Employee *employee) {  
    printf("Employee info: \n");  
    printf("\tFIO: %s\n", (*employee).fio);  
    printf("\tPosition: %s\n", (*employee).position);  
    printf("\tSalary: %.2f\n", (*employee).salary);  
}
```

Output:
Employee info:
 FIO: Ivanov Ivan Ivanovich
 Position: Developer
 Salary: 100500.00

Employee info:
 FIO: Petrov Petr
 Position: QA
 Salary: 2000.00

Note: don't do such things:

typedef struct Student Student;

or

typedef struct Student * Student_t;

<https://yarchive.net/comp/linux/typedefs.html>

Composite structures

```
/* note, forward declaration is  
allowed as well */
```

```
struct Date {  
    uint16_t year;  
    uint8_t month;  
    uint8_t day;  
};
```

```
struct Employee {  
    char fio[50];  
    double salary;  
    char position[20];  
    struct Date dob;  
};
```

```
struct Employee ivanov;  
strcpy(ivanov.fio, "Ivanov Ivan Ivanovich");  
strcpy(ivanov.position, "Developer");  
ivanov.salary = 100500;  
ivanov.dob.day = 20;  
ivanov.dob.month = 11;  
ivanov.dob.year = 2001;
```

```
/* NOTE: order of fields is strict!!! */  
struct Employee petrov = { "Petrov Petr", 2000, "QA", { 1985,  
10, 30} };
```

Enumerations

```
enum Position {  
    DEV, QA, PM, BA  
};
```

```
struct Employee {  
    char fio[50];  
    double salary;  
    enum Position position;  
    struct Date dob;  
};
```

```
struct Employee ivanov;  
strcpy(ivanov.fio, "Ivanov Ivan");  
ivanov.position = DEV;
```

```
printf("%s", ivanov.position);
```

Will not work with warning: Format specifies type 'char *' but the argument has underlying type 'unsigned int'

Case 1:

```
switch ((*employee).position) {  
    case DEV: printf("\tPosition: Developer\n"); break;  
    case QA: printf("\tPosition: Quality Assurance\n"); break;  
    /* ... */  
}
```

Case 2:

```
enum Position { DEV, QA, PM, BA };  
static char positions[4][20] = { "Developer", "Quality Assurance",  
    "Project manager", "Business Analytic" };  
printf("\tPosition: %s\n", positions[(*employee).position]);
```

Enumeration list

- Sample 1: make employee take several positions at once (e.g., Dev & TeamLead, QC & ScrumMaster, Dev & PM & QC (freelancer ;))
- Sample 2: (real) Linux FileSystem's file attributes (R – 4, W – 2, X – 1).
 - 0xxx -> Orwx -> 8421
 - Byte = 5 => -r-x
- Implementation 1:
 - ```
struct Employee {
 enum Position positions[10];
}; /* why all employees should have such redundancy ? */
```
- Implementation 2:
  - ```
struct Employee {  
    enum Position* positions;  
}; /* too messy to handle this in C, and time/memory consumption */
```

Enumeration List. Impl #3

- Preface: all values in enumerations are digits. Why not to use bitwise operators?
- Pre-code:
 - enum Position { *DEV* = 1, *QA* = 2, *PM* = 4, *BA* = 8 };
 - $7 \Rightarrow 1+2+4 \Rightarrow \text{DEV} | \text{QA} | \text{PM} \Rightarrow 0111$
 - struct Employee {
 unsigned int positions;
};
 - employee.positions = *BA*;
 - employee.positions = *BA* | *DEV*;

addPosition:

Init = 1000 => BA

Set = 0001 -> Dev

Operation - |

Result = 1001

clearPosition:

Init = 1010 => BA | QA

Clear = 0010 -> QA

Operation - ?

Result = 1000

hasPosition:

Init = 1010 => BA | QA

Has? = 0010 -> QA

Operation - ?

Result = ?

Enumeration List. Code

```
void setPosition(struct Employee *employee, enum Position pos) {    void printPosition(int positions) {  
    employee->positions |= pos;  
}  
  
void clearPosition(struct Employee *employee, enum Position pos)  
{  
    employee->positions &= ~pos;  
}  
  
bool hasPosition(int positions, enum Position pos) {  
    return positions & pos;  
}  
  
    printf("\tPositions:%d ", positions);  
    for (int i = 0; i < 4; i++) { // 4 - count of elements in enum  
        if (hasPosition(positions, (int)pow(2, i))) { // 1<<i  
            printf("%s, ", positionsText[i]);  
        }  
    }  
    printf("\n");  
}
```

Structures arrays

```
#define EMPLOYEE_COUNT 10
struct Employee employees[EMPLOYEE_COUNT];
for (int i = 0; i < EMPLOYEE_COUNT; i++) {
    employees[i].salary = rand()%5000 + 2000;
    employees[i].position = rand()%4;
    // for fio we can use set of fnames-mnames-lnames, rand them
    // and concat together; or make concat of 'Employee #' + string
interpret
    // of random/consequent integer
}
```

Structures array

```
// sort by salary DESC
for (int i = 0; i < EMPLOYEE_COUNT; i++) {
    for (int j = 0; j < EMPLOYEE_COUNT - 1; j++) {
        if (employees[j].salary < employees[j + 1].salary) {
            struct Employee temp = employees[j];
            employees[j] = employees[j+1];
            employees[j+1] = temp;
        }
    }
}

for (int i = 0; i < EMPLOYEE_COUNT; i++) {
    printEmployee(&employees[i]);
}
```

Struct array sort

```
int cmpfunc (const void * a, const void * b){  
    return ( (*(struct Employee*)a).salary - (*(struct Employee*)b).salary );  
}  
  
int main(void) {  
  
    struct Employee employees[] = {  
        {"ivanov", 123, DEV },  
        {"petrov", 423, QA},  
        {"sidorof", 111, PM}  
    };  
  
    qsort(employees, 3, sizeof(struct Employee), cmpfunc);  
  
    return 0;  
}
```

Dynamic structures

```
struct Employee * employee = malloc(1 * sizeof(struct Employee));
(*employee).salary=100500; // employee[0].positions=BA;
employee->salary=100500;
strcpy(employee->fio, "Some FIO");
free(employee);

// better use `struct Employee **` ;
struct Employee * employees = malloc(10 * sizeof(struct Employee));
*(employees + 4).salary=2000;
free(employees);
```

Note: naming convention

Structures storage in file (1/4)

Beginner level: One line in file – one field value

```
ivanov ivan
123.000000
Dev
petrov petr
423.000000
QA
sidorof sidr
111.000000
PM
```

```
void writeToFile(struct Employee* emp, FILE* fd) {
    fprintf(fd, "%s\n", emp->fio);
    fprintf(fd, "%f\n", emp->salary);
    fprintf(fd, "%s\n", positions[emp->position]);
}

FILE *file = fopen("emp.txt", "w");
for (int i = 0; i < 3; i++) {
    writeToFile(&employees[i], file);
}
fclose(file);
```

Structures storage in file (1/4)

Beginner level: One line in file – one field value

ivanov ivan	
123.000000	
Dev	
petrov petr	
423.000000	
QA	
sidorof sidr	
111.000000	
PM	

```
void readFromFile(struct Employee* emp, FILE* fd) {  
    fgets(emp->fio, 50, fd); // read full line  
    fscanf(fd, "%lf\n", &emp->salary);  
    char pos[10];  
    fscanf(fd, "%s\n", pos);  
    emp->position = 0; // TODO: convert to pos to position: emp->position = ... ;  
}  
FILE *file = fopen("emp.txt", "r");  
struct Employee readEmp[10];  
size_t readCount = 0;  
while (!feof(file)) {  
    readFromFile(&readEmp[readCount++], file);  
}  
fclose(file);
```

Structures storage in file (2/4)

Intermediate level: One line in file – one record

```
void readFromFile(struct Employee* emp, FILE* fd) {  
    char buffer[100];  
    char * x = fgets(buffer, 100, fd); // read full line and parse it  
    if (x == NULL) return; // in case we have last empty line (!!!)  
  
    char * token = strtok(buffer, ",");  
    strcpy(emp->fio, token);  
    token = strtok(0, ",");  
    emp->salary = strtod(token, NULL);  
    token = strtok(0, ","); // pos., e.g. "Dev\n"  
    emp->position = 0; // TODO: convert to pos to position: emp->position = ... ;  
}
```

```
void writeToFile(struct Employee* emp, FILE* fd) {  
    fprintf(fd, "%s,%f,%s\n", emp->fio, emp->salary,  
    positions[emp->position]);  
}
```

ivanov ivan,123.000000,Dev
petrov petr,423.000000,QA
sidorof sidr,111.000000,PM

Structures storage in file (3/4): Advanced Level: binary storage / serialization

- `FILE *file = fopen("emp2.txt", "w");`
`fwrite(employees, sizeof(struct Employee), 3, fd);`
`fclose(file);`
- `FILE *file = fopen("emp2.txt", "r");`
`struct Employee readEmp[10];`
`fread(readEmp, sizeof(struct Employee), 3, fd);`
`fclose(file);`
- // count of employees: filesize / sizeof(struct Employee)

Structures storage in file (3/4): Advanced Level: binary storage / serialization

- `hexdump -C emp2.txt`

```
00000000  69 76 61 6e 6f 76 20 69  76 61 6e 00 00 00 00 00 |ivanov ivan.....|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |.....|
*
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 c0 5e 40 |.....□^@|
00000040  00 00 00 00 00 00 00 00  70 65 74 72 6f 76 20 70 |.....petrov p|
00000050  65 74 72 00 00 00 00 00  00 00 00 00 00 00 00 00 |etr.....|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |.....|
*
00000080  00 00 00 00 00 70 7a 40  01 00 00 00 00 00 00 00 |.....pz@.....|
00000090  73 69 64 6f 72 6f 66 20  73 69 64 72 00 00 00 00 |sidorof sidr....|
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |.....|
*
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 c0 5b 40 |.....□[@|
000000d0  02 00 00 00 00 00 00 00
```

Structures storage in file (4/4)

- Pro Level: JSON / XML

```
[  
 {  
     "fio": "ivanov ivan",  
     "salary": 123.000000,  
     "position": "Dev"  
 }, {  
     "fio": "petrov petr",  
     "salary": 423.000000,  
     "position": "QA"  
 }, {  
     "fio": "sidorof sidr",  
     "salary": 111.000000,  
     "position": "PM"  
 }]  
 ]
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<workers>  
    <worker>  
        <fio>ivanov ivan</fio>  
        <position>Dev</position>  
        <salary>123.0</salary>  
    </worker>  
    <worker>  
        <fio>petrov petr</fio>  
        <position>QA</position>  
        <salary>423.0</salary>  
    </worker>  
    <worker>  
        <fio>sidorof sidr</fio>  
        <position>PM</position>  
        <salary>111.0</salary>  
    </worker>  
</workers>
```

Unit tests with structures

```
enum Position { DEV, QC, BA, PM };
struct Employee {
    char fio[20];
    double salary;
    enum Position position;
};
struct Employee* max_salary_employee(struct Employee* employees, size_t size);

void assertEmployee(struct Employee* employeeActual, struct Employee* employeeExpected) {
    ck_assert_str_eq(employeeActual->fio, employeeExpected->fio);
    ck_assert_int_eq(employeeActual->position, employeeExpected->position);
    ck_assert_double_eq_tol(employeeActual->salary, employeeExpected->salary, 0.01);
}

START_TEST(test_max_salary_employee)
{
    struct Employee employees[3] = {
        {"Ivanov", 123.45, PM},
        {"Petrov", 432.45, QC},
        {"Sidorov", 125.45, DEV},
    };
    struct Employee expectedEmployee = {"Sidorov", 125.45, DEV};
    struct Employee* actualEmployee = max_salary_employee(employees, 3);
    assertEmployee(actualEmployee, &expectedEmployee);
}
END_TEST
```

Other

- Single rule: *NEVER* pass (and return) structures ‘by value’ into the function
 - Memory consumption
 - Remember that we’re passing copy of the structure, thus its modification is not a case
- Make copy of structure:
 - struct Employee sidorov = ivanov;
- Enumeration values are global, so make sure its uniqueness
- enum Position { *DEV* = 5, *QA*, *PM*, *BA* }; /* *DEV* = 5, *QA* = 6, *PM* = 7 .. */

Other. Bit Fields

Structure Bit fields used to compact data, but in real world its odd. In addition, default alignment is 2 bytes. Sample of code:

```
struct Date {  
    uint8_t day: 5;  
    uint8_t month: 4;  
    uint16_t year: 11;  
};
```

Should use 3 bytes.

Other. Unions

- Goal: save memory (if we don't need all the values at once)
- Union's size – size of the 'heaviest' field
- All the fields share same memory
- Sample
 - ```
union FooUnion {
 char text[6];
 int value;
};
```
  - ```
union FooUnion f;  
f.value = 100500; // f.text -> \x94\x88\x01 ;
```
 - note: 100500 in hex = 0x018894

Dynamic Data Structures

- Dynamic Arrays
- Linked lists
- Reason:
 - We can't modify arrays dimension “on the fly”
 - We have no ability to use spared array of infinity size

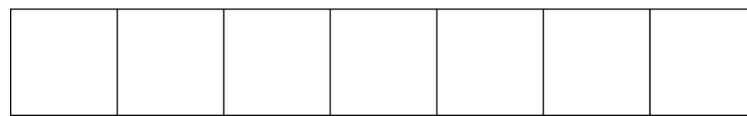
How to process dynamic arrays (practically)

- Create data model:
 - ```
struct TestElement {
 int id; // ideally, domain element should contain some unique data, e.g. fio, inn
 char data[30];
};
```
- Create List representation (e.g. `group` for list of students, `warehouse` for list of goods)
  - ```
struct Container {  
    size_t size;  
    struct TestElement** data; // NOTE: array of pointers  
};
```
- Create functions to operate with list:
 - `void insert(struct Container *array, int pos, <data>)`
 - `void delete (struct Container *array, int pos)`
 - `struct TestElement* findByCriteria(<criteria>)` // return pointer as it may show absence of result
 - `void print(struct Container *array) + void print(struct TestElement *array)`

Dynamic arrays. Insert

5	4	10	12	7	8
---	---	----	----	---	---

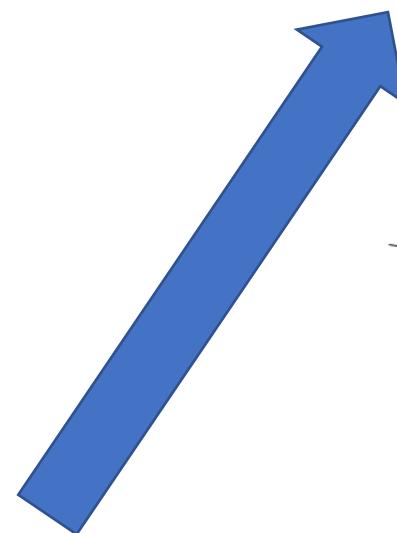
size = 6
insert pos = 3
insert value = 20



5	4	10	12	7	8
---	---	----	----	---	---

size = 6
insert pos = 3
insert value = 20

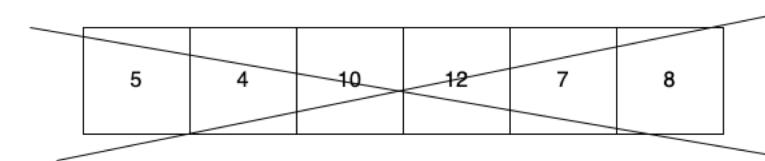
5	4	10	20			
---	---	----	----	--	--	--



5	4	10	12	7	8
---	---	----	----	---	---

size = 6
insert pos = 3
insert value = 20

5	4	10	20	12	7	8
---	---	----	----	----	---	---



size = 6
insert pos = 3
insert value = 20

5	4	10	20	12	7	8
---	---	----	----	----	---	---

Dynamic arrays. Insert. Code

```
void insert(struct Container* container, size_t pos, struct TestElement* element) {
    struct TestElement** new_array = malloc((container->size + 1) * sizeof(struct TestElement*));

    if (pos > container->size) pos = container->size;

    memcpy(new_array, container->array, pos * sizeof(struct TestElement*));
    memcpy(new_array + pos, element, sizeof(struct TestElement*));
    memcpy(new_array + pos + 1, container->array + pos, (container->size - pos) * sizeof(struct TestElement*));

    free(container->array); // don't free each element

    container->array = new_array;
    container->size++;
}
```

Dynamic arrays. Remove

5	4	10	20	12	7	8
---	---	----	-----------	----	---	---

size = 7
remove pos = 3

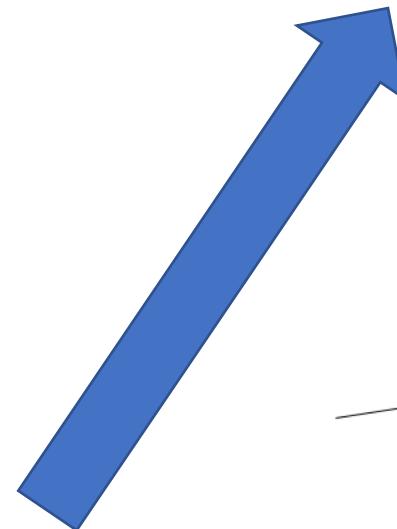
--	--	--	--	--	--	--



5	4	10	20	12	7	8
---	---	----	-----------	----	---	---

size = 7
remove pos = 3

5	4	10				
---	---	----	--	--	--	--



5	4	10	20	12	7	8
---	---	----	-----------	----	---	---

size = 7
remove pos = 3

5	4	10	12	7	8
---	---	----	-----------	---	---



5	4	10	20	12	7	8
---	---	----	---------------	----	---	---

size = 7
remove pos = 3

5	4	10	12	7	8
---	---	----	----	---	---

Dynamic arrays. Delete. Code

```
void delete(struct Container * container, size_t pos) {
    if (container->size == 0) return; // nothing to delete
    struct TestElement** new_array = malloc((container->size - 1) * sizeof(struct TestElement*));

    if (pos >= container->size) pos = container->size - 1;
    free(container->array[pos]);

    memcpy(new_array, container->array, pos * sizeof(struct TestElement*));
    memcpy(new_array + pos, container->array + pos + 1, (container->size - pos - 1) * sizeof(struct TestElement*));

    free(container->array);
    container->array = new_array;
    container->size--;
}
```