МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет “Львівська політехніка”

ДИНАМІЧНЕ ВИДІЛЕННЯ ПАМ’ЯТІ

ІНСТРУКЦІЯ
до лабораторної роботи № 9 з курсу
“Основи програмування”
для базового напрямку “Програмна інженерія”

Затверджено
На засіданні кафедри
програмного забезпечення
Протокол № від

ЛЬВІВ – 2018

1. МЕТА РОБОТИ

Мета роботи – навчитися використовувати динамічну пам'ять, виділяти та звільняти її засобами мови С.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Динамічні змінні

Бувають ситуації, коли заздалегідь невідомо, скільки даних потрібно обробити у програмі. Особливо це стосується роботи з масивами – при оголошенні статичного масиву для нього виділяється жорстко фіксована пам'ять. Однак, подекуди ця жорстко фіксована кількість елементів масиву може перевищувати реальну кількість даних або ж навпаки – бути недостатньою. Якщо масив не може вмістити потрібну кількість даних, доведеться переписувати і повторно компілювати програму. Один з типових підходів при використанні статичних масивів – виділити пам'ять з запасом, а заповнити даними лише частину масиву і в якійсь змінній зберігати фактичну кількість елементів масиву. У цьому випадку виділена пам'ять використовується неефективно. Крім того, статичні змінні "живуть" визначений час – або до кінця роботи програми, або до кінця її певного блока. Взагалі кажучи, часто статична змінна вже не є потрібною, а все ще займає пам'ять.

У мові С існує можливість динамічно розподіляти пам'ять у ході виконання програми, тобто, виділяти стільки пам'яті, скільки її насправді потрібно, і звільнити її тоді, коли вона вже не буде потрібною. Пам'ять виділяється з так званої вільної пам'яті. Розмір вільної пам'яті залежить від операційної системи та моделі пам'яті комп'ютера. Динамічні змінні не мають імен, а доступ до них здійснюється через вказівники. Динамічні об'єкти розміщаються у "**кучі**" (англомовний термін-відповідник: **heap**). Для управління вільною пам'яттю у С використовується набір функцій **malloc**(), **calloc**(), **realloc**() та **free**().

2.2 Використання функцій malloc(), calloc(), realloc() та free()

Функція
`void *malloc(size_t size);`
виділяє в динамічній пам'яті блок розміром `size` байт. Виділена пам'ять зайнята довільними даними – "сміттям". Якщо пам'ять виділена успішно, то `malloc` повертає нетипізований вказівник на початок виділеного блока пам'яті, а інакше `NULL`.

Наприклад:

```
	char* p = (char*)malloc(sizeof(char)); /* створення динамічної змінної розміром, визначеним типом char, тобто, розміром в 1 байт  */
	int* p = (int*)malloc(sizeof(int)); /* створення динамічної змінної розміром, визначеним типом int  */
	char* p = (char*)malloc(100*sizeof(char)); /* створення масиву розміром у 100 змінних типу char */
	int* p = (int*)malloc(100*sizeof(int)); /* створення масиву розміром 100 змінних типу int */

	struct student{
		char surname[50];
		char name[50];
		int mark;
	};
	typedef struct student TStudent;
	int k = 10;
	TStudent * p = (TStudent*)malloc(k*sizeof(TStudent)); /* створення масиву розміром у k змінних типу student; k є саме змінною, а не константою, що було б недопустимо при створенні статичного масиву */
```

Ключове слово `typedef` використовуються для створення типу-синоніма структурі `student`. При цьому використання типу `TStudent` рівнозначне використанню типу `struct student` (в цьому випадку використовується для скорочення коду).

Приклад. В наступній програмі користувача запрошують ввести кількість елементів масиву, в динамічній пам'яті створюється масив вказаного розміру, який заповнюється даними, введеними з клавіатури, що згодом виводяться на екран у зворотному порядку.

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

int main()
{
	int n;
	printf("Enter number of elements in the array: ");
	scanf("%d", &n);

	int* arr = (int*)malloc(n*sizeof(int));
	if ( !arr )
		printf("Error allocation %d integer items.\n", n);
	else
	{
		int* p = arr; /* за допомогою вказівника p будемо переміщатися
по динамічному масиву, тоді як вказівник arr лишимо незмінним – він буде
вказувати на початок блоку пам'яті */
		int i;
		for (i=0; i < n; ++i, ++p)
		{
			printf("Enter %d-th item: ", i);
			scanf("%d", p);
		}

		p = arr + n - 1; /* встановлюємо вказівник p на останній елемент
масиву та у циклі виводимо на екран всі n елементів масиву*/
		for (i=n; i>0; --i, --p)
			printf("%d\n", *p);

		free(arr);
	}

	printf("Press any key to exit.");
	_getch();

	return 0;
}
```

З динамічним масивом можна працювати які зі звичайним масивом – через індекси. Та ж сама програма з використанням доступу до елементів динамічного масиву через індекси:

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

int main()
{
	int n, i;
	printf("Enter number of elements in the array: ");
```

```
	scanf("%d", &n);

	int* arr = (int*)malloc(n*sizeof(int));
	if ( !arr )
		printf("Error allocation %d integer items.\n", n);
	else
	{
		for (i=0; i < n; ++i)
		{
			printf("Enter %d-th item: ", i);
			scanf("%d", &arr[i]);
		}

		for (i=n-1; i>=0; --i)
			printf("%d\n", arr[i]);

		free(arr);
	}

	printf("Press any key to exit.");
	_getch();

	return 0;
}
```

Функція
`void *calloc(size_t numb, size_t size);`
виділяє в динамічній пам'яті суцільний масив з `numb` елементів, кожен з яких займає `size` байтів. Виділений блок пам'яті заповнюється нулями. Якщо пам'ять виділена успішно, повертається вказівник на початок виділеного блока, а інакше значення `NULL`.

Функція
`void *realloc(void *ptr, size_t size);`
встановлює у `size` розмір блока пам'яті, виділеного раніше функціями `malloc`, `calloc` або `realloc`, на початок якого вказує `ptr`.
Якщо `ptr == NULL`, то функція `realloc` діє аналогічно функції `malloc` з параметром `size`.
Якщо зміна розміру блока пам'яті відбулася успішно, то функція `realloc` повертає вказівник на початок зміненого блока (у разі, якщо адреса блока міняється, дані копіюються в ділянку пам'яті за новою адресою). Адреса блока може і не змінитися. У будь-якому випадку за старою адресою до даних звертатися не можна – потрібно перевірити, чи адреса не змінилася. Якщо ж розмір блока змінити не вдалося, то функція `realloc` повертає `NULL`, а до даних можна продовжувати звертатися за старою адресою.

Функція
`void free(void *ptr);`
звільняє блок пам'яті, на початок якого вказує `ptr`. Добрим стилем є звільнення пам'яті щоразу, як вона стає непотрібною. Після завершення роботи програми пам'ять буде звільнено, тому в невеликих програмах звільнення пам'яті є питанням доброго стилю. Однак, у великих програмах є небезпека «просочування пам'яті».

Приклад. Наступна програма приймає від користувача цілі числа та записує їх у динамічний масив. Спочатку виділяється місце під N елементів типу int. Далі, якщо кількість введених чисел перевищить N, то виділяємо вдвічі більший обсяг пам'яті. Програма працюватиме

коректно, якщо користувач задасть не більше, ніж 2N цілих чисел. Ввід чисел буде припинено, якщо користувач задасть 0.

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

int main()
{
	int N = 6; /* спочатку виділяємо пам'ять для збереження N цілих чисел; оскільки calloc повертає вказівник на void, а працювати потрібно з int, то слід явно привести результат calloc до типу int* */
	int *p = (int*)calloc(N, sizeof(int));
	if ( p ) /* якщо пам'ять виділена успішно */
	{
		int *pp = p, *p2 = NULL; /* оголошуємо «робочі» вказівники */
		int i = -1, k = -1, j; /* i - будемо використовувати як індекс в масиві, а k, j - допоміжні змінні */
		do { /* поки користувач вводить ненульові числа, їх будемо записувати в динамічний масив */
			i++; k++;
			if ( k > (N-1) ) /* якщо введено більше, ніж N чисел */
			{	/* збільшуємо розмір динамічного масиву вдвічі*/
				p2 = (int*)realloc(p, 2*N*sizeof(int));
				if ( !p2 ) /* якщо не вдалося виділити пам'ять, то виходимо з циклу */
				{
					i--;
					printf("Sorry, not enough memory");
					break;
				}
				else
				{
					/* допоміжну змінну k знову встановлюємо в її початкове значення і встановлюємо «робочий» вказівник pp на нову адресу динамічного масиву */
					pp = p2;
					k = -1;
				}
			}

			printf("Enter a number ");
			scanf("%d", &pp[i]);

		} while (pp[i]);

		/* Якщо p2!=NULL, то виділялася додаткова пам'ять, і тоді «робочому» вказівнику pp присвоюємо значення p2, а інакше - значення p (тобто, вказівника на початок блока пам'яті, що була виділена від самого початку функцією calloc)*/
		pp = p2 ? p2 : p;
		/* вивід елементів динамічного масиву на екран */
		for (j = 0; j < i; j++)
			printf("%d\t", pp[j]);
	}

	_getch();
	return 0;
}
```

Приклад. В наступній програмі передбачається заповнення полів екземплярів структур даними, зчитаними з текстового файлу (один рядок файлу містить дані для одного екземпляра структури, розділені знаками табуляції). Спочатку зчитується кількість рядків у файлі (файл відкривається для читання і зчитується по рядках, причому кожного разу спеціально оголошена змінна збільшується на 1). Після завершення читання файл закривається. Тоді в динамічній пам'яті виділяється область для зберігання стількох екземплярів структур, скільки рядків знайдено у текстовому файлі. Файл знову відкривається для читання – у циклі `while` поки не досягнуто кінець файлу на кожній ітерації зчитується один рядок. Кожен зчитаний рядок розділяється на окремі дані за допомогою функції `strtok`. Відокремлені дані записуються на кожній ітерації у відповідні поля поточного екземпляра структури у динамічній пам’яті (там, де це потрібно, виконується приведення типів).

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <conio.h>

#define STUDSTRLEN 50
#define FILESTRLEN 200
#define FILENAME "student.txt"

int main()
{
	struct student
	{
		char surname[STUDSTRLEN];
		char name[STUDSTRLEN];
		int mark;
	};
	typedef struct student TStudent;

	char strfromfile[FILESTRLEN]; /* у цю змінну будемо зчитувати вміст
файлу по рядках */

	FILE* f = fopen(FILENAME, "r");
	if ( !f )
	{
		int lasterror = errno;
		printf("Error opening file \"%s\". %s\n", FILENAME,
strerror(lasterror));
		return lasterror;
	}

	int k = 0; /* у змінній k зберігатимемо кількість рядків у текстовому
файлі student.txt */
	while( !feof(f) )
	{
		fgets(strfromfile, FILESTRLEN, f);
		k++;
	}
	fclose(f); /* закриваємо файл */

	TStudent* pp = (TStudent*)malloc(k*sizeof(TStudent));/* виділяємо
рівно стільки вільної пам’яті, скільки потрібно для створення масиву
екземплярів структури student */
	if ( pp ) /* якщо пам'ять була успішно виділена */
	{
```

```
            TStudent* p = pp;
            int i = 1;

            f = fopen(FILENAME, "r"); /* знову відкриваємо файл student.txt
для читання */
            if ( !f )
            {
                  int lasterror = errno;
                  free(pp); /* звільняємо попередньо виділену пам'ять */
                  printf("Error opening file \"%s\". %s\n", FILENAME,
strerror(lasterror));
                  return lasterror;
            }

            char *pw; /* вказівник для розділення зчитаних рядків на окремі
дані */

            while( !feof(f) ) /* поки не досягнутий кінець файлу */
            {
                  fgets(strfromfile, FILESTRLEN, f); /* читаємо поточний
рядок у змінну strfromfile, припускаючи, що довжина кожного рядка файлу не
перевищує 200 символів; на кожній ітерації циклу курсор посувається на одну
позицію */

                  int j = 0;
                  pw = strtok(strfromfile, "\t");  /* дані розділяються за
символами табуляції */
                  while( pw )
                  {
                        switch( j ) /* на кожній ітерації циклу while зі
зчитаного текстового рядка виділяється одна певна частина даних, яка
записується у відповідне поле структури */
                        {
                              case 0: strcpy(p->surname, pw); break;
                              case 1: strcpy(p->name, pw); break;
                              case 2: p->mark=atoi(pw); break;
                        }
                        pw = strtok (NULL, "\t");
                        j++;
                  }
                  ++p;
            }
            fclose(f);

            /* вивід полів усіх структур на екран; поля однієї структури
розділяються знаками табуляції, а різні екземпляри структури записуються в
різних рядках  */
            for (p=pp; p!=pp+k; p++)
                  printf("%s\t%s\t%d\n", p->surname, p->name, p->mark);

            free(pp);
      }

      _getch();
      return 0;
}
```

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. У чому полягає суть динамічного виділення пам’яті?
2. Як називається область оперативної пам’яті, яка призначена для динамічного виділення?
3. Як динамічно виділити пам’ять засобами мови С?
4. Яка різниця між функціями calloc та malloc?
5. Для чого потрібно звільняти динамічну пам’ять? Як це можна зробити в С?

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Ознайомитися з теоретичним матеріалом викладеним вище в даній інструкції і виконати приклади програм.
2. Одержати індивідуальне завдання.
3. Скласти програму на мові С++ у відповідності з розробленим алгоритмом.
4. Виконати обчислення по програмі.
5. Підготувати та здати звіт про виконання лабораторної роботи.

5. СПИСОК ЛІТЕРАТУРИ

1. Керниган Б., Ритчи Д. Язык программирования С. - М. - Финансы и статистика. - 1992. – 272 с.
2. Уэйт М., Прата С., Мартин Д. Язык С. Руководство для начинающих. - М. - Мир. - 1988. – 512 с.
3. К. Джамса. Учимся программировать на языке С++. М.: Мир, 1997. – 320 с
4. Герберт Шилдт. Полный справочник по С++. М. – С.-П.-К., Вильямс. – 2003. – 800 с.
5. Демидович Е. М. Основы алгоритмизации и программирования. Язык Си. (Учебное пособие). – Санкт-Петербург: “БХВ Петербург”. – 2006. – 439 с.

6. ІНДИВІДУАЛЬНІ ЗАВДАННЯ

1. Ввести розмір квадратної матриці і її елементи. Елементи матриці розташувати в динамічній пам'яті. Визначити номер стовпця, у якого сума елементів, розташованих вище головної діагоналі, максимальна.
2. Ввести матрицю з кількістю рядків k . Рядки матриці мають змінну довжину, елементи матриці розмістити в динамічній пам'яті. Обчислити й зберегти суму елементів кожного рядка, а потім вивести їх на екран.
3. Ввести розмір квадратної матриці і її елементи. Елементи матриці розташувати в динамічній пам'яті. Визначити номер стовпця, у якого сума елементів, розташованих нижче головної діагоналі найменша. Обнулити елементи, використовувані при підрахунку цих сум.
4. Ввести num - кількість масивів. Ввести розмірність чергового масиву і його елементи цілого типу, розмістити їх у динамічній пам’яті. Розсортувати масиви по зростанню і вивести на екран.
5. Написати програму для об'єднання масивів, n – кількість масивів, що підлягають об'єднанню; a -масив вказівників на масиви, що підлягають об'єднанню; масив size -містить розміри масивів, що підлягають об'єднанню. Пам'ять під масиви виділити динамічно; totar – результуючий масив..
6. Ввести двохмірний масив, для роботи з масивом використати вказівник. Масив розмістити в динамічній пам'яті. Визначення суми елементів вказаного рядка масиву.
7. Ввести не більше 5 масивів цілих чисел. Кількість чисел у масиві задається під час виконання програми, це число записати першим елементом масиву. Масиви розмістити в динамічній пам'яті. Створити масив вказівників на дані масиви. У функції вивести на екран всі елементи кожного масиву.
8. Ввести num - кількість масивів. Ввести розмірність чергового масиву і його елементи цілого типу, розмістити їх у динамічній пам’яті. Розсортувати масиви по спаданню і вивести у файл.

9. Ввести не більше 3 масивів цілих чисел. Кількість чисел у масиві задається під час виконання програми, це число записати першим елементом масиву. Масиви розмістити в динамічній пам'яті. Створити масив вказівників на дані масиви. У функції обчислити суму елементів всіх масивів.
10. Використати динамічне виділення пам’яті для двомірного масиву цілих чисел, розмірності масиву ввести з клавіатури. Для роботи з масивом використати вказівник на вказівник. Роздрукувати елементи масиву.
11. Ввести разміри матриці n, m, розмістити матрицю в динамічній пам'яті. Для роботи з елементами матриці використати звичайний вказівник. Вивести елементи 2-ого стовпчика матриці на екран.
12. Ввести розмір квадратної матриці і її елементи. Елементи матриці розташувати в динамічній пам'яті. Визначити чи є стовпці з одинаковими сумами елементів.
13. Ввести матрицю з кількістю рядків k . Рядки матриці мають змінну довжину, елементи матриці розмістити в динамічній пам'яті. Обчислити й зберегти суми елементів 1-ого і останнього рядків, і вивести їх на екран.
14. Ввести розмір квадратної матриці і її елементи. Елементи матриці розташувати в динамічній пам'яті. Визначити номер стовпця, у якого сума елементів, розташованих нижче головної діагоналі найбільша.
15. Ввести n - кількість масивів. Ввести розмірність чергового масиву і його елементи типу float, розмістити їх у динамічній пам’яті. Розсортувати масиви по зростанню і записати у файл.
16. Написати програму для об'єднання декількох масивів, k – кількість масивів, що підлягають об'єднанню; s -масив вказівників на масиви, що підлягають об'єднанню; масив dim -містить розміри масивів, що підлягають об'єднанню. Пам'ять під масиви виділити динамічно; arra – результуючий масив..
17. Ввести двохмірний масив, для роботи з масивом використати вказівник. Масив розмістити в динамічній пам'яті. Визначити суми елементів вказаного стовпчика масиву.
18. Ввести не більше 3 масивів чисел з плаваючою крапкою. Кількість чисел у масиві задається під час виконання програми введенням з клавіатури. Масиви розмістити в динамічній пам'яті. Створити масив вказівників на дані масиви. У функції вивести на екран всі елементи кожного масиву.
19. Ввести num - кількість масивів. Ввести розмірність чергового масиву і його елементи типу double, розмістити їх у динамічній пам’яті. Розсортувати масиви по спаданню і вивести на екран.
20. Ввести не більше 4 масивів цілих чисел. Кількість чисел у масиві задається під час виконання програми, це число записати першим елементом масиву. Масиви розмістити в динамічній пам'яті. Створити масив вказівників на дані масиви. У функції обчислити суму елементів всіх масивів.
21. Ввести двохмірний масив, для роботи з масивом використати вказівник. Масив розмістити в динамічній пам'яті. Визначення мінімальний елемент вказаного стовпчика.
22. Написати програму для об'єднання декількох масивів, m – кількість масивів, що підлягають об'єднанню; s -масив вказівників на масиви; масив dim -містить розміри даних масивів. Пам'ять під масиви виділити динамічно; masyv – результуючий масив.
23. Створити структуру з прізвищами студентів та їх іменами. Розсортувати за алфавітом список по прізвищу, переміщаючи вказівники на записи. Вивести отриманий список на екран. Інформацію розмістити в динамічній пам'яті.