Національний технічний університет України
«Київський політехнічний інститут»

Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп'ютерних систем

ЛАБОРАТОРНА РОБОТА №2

з дисципліни

«Паралельні та розподілені обчислення"

ТЕМА: «Робота з компіляторами мов C та Java в режимі командного рядка»

Підготував: доц. Марченко О.І.

Київ – 2009-2016

Основи роботи з gcc

Основним компілятором для **ОС Linux** є **gcc** (gnu c compiler). Він входить до колекції компіляторів **GCC** (GNU Compilers Collection). Для перевірки чи встановлений компілятор необхідно виконати наступну команду:

```
$ gcc
```

Якщо у відповідь буде відображено:
`gcc: no input files`
то компілятор встановлений і готовий до роботи.

Основні опції **gcc**:

-OX — де X задає рівень оптимізації компілятора від 0 до 3. Чим більше число тим складніші оптимізації використовуються. При налагодженні програми необхідно використовувати рівень 0. Після того як програма була налагоджена рекомендується використовувати рівень 2. Винятком є значення s (-Os), яка вказує що необхідно оптимізувати за розміром. Така оптимізація може використовуватися для вбудованих систем у який бракує пам'яті. За замовченням використовується рівень 2;

-o <file_name> - ім'я файлу що буде отримано після компіляції (програми, об'єктного файлу чи бібліотеки);

-c — вказує на те що компілюється об'єктний файл;

-g — вказує що необхідно компілювати програму із інформацією для налагодження.

Компіляція та запуск створеної програми

Найпростіший випадок компіляції може бути представлено наступним чином:
`gcc -o <ім'я файлу результату> <опції компіляції> <список вихідних та об'єктних файлів для компіляції>`

Збережіть наступний фрагмент вихідного коду у файлі “hello.c”. У кінці файлу рекомендується ставити порожній рядок:

```
#include <stdio.h>
int main()
{
    printf(“Hello, world!\n”);
    return 0;
}
```

Компілювати цю програму можливо так:

```
$ gcc -o hello hello.c
```

Для виконання програми hello:

```
$ ./hello
```

Компіляція складних програм на мові С і використання make

Утиліта **make** використовується для автоматизації процесу компіляції програми. За допомого спеціального командної мови можна прописати набір правил для компіляції програми. Усі правила повинні бути збережені у файлі з назвою Makefile.

Правило має наступну структуру:

```
<імена цільових файлів> : <імена файлів реквізитів>
<tab><tab><команда 1>
<tab><tab><команда 2>
...
<tab><tab><команда N>
```

<імена цільових файлів> - імена файлів, що будуть отримані в результаті роботи команд. Винятком є вказування замість імені файлу певної дії, яка буде виконана у результаті роботи команд;

<імена файлів реквізитів> - імена файлів від яких залежать цільові файли (або дії);

<команда X> - це команда яку необхідно виконати. Команди виконуються згори вниз. Рядки, в яких записано команди повинні починатися з символу табуляції.

Змінні оточення, що можуть бути використані утилітою make:

CC — змінна, що зберігає ім'я компілятору мови С що використовується. Наприклад,

CC=gcc

вказує на те, що потрібно використовувати компілятор gcc;

CFLAGS — змінна, що вказує, які прапорці (опції) компіляції повинні використовуватись при компіляції програми. Наприклад,

CFLAGS=-O2

вказує, що необхідно використати оптимізацію другого рівня.

Повний список змінних, що можуть бути використані, може бути знайдений у документації до утиліти **make**.

Обробка списку аргументів, що передаються програмі на мові С (argc, argv[]).

Для того, щоб передати компілятору інформацію, що дана програма при запуску на виконання може отримувати параметри (аналогічно до того, як отримують параметри функції), при написанні головної функції main цієї програми необхідно у її заголовку описати два спеціальних аргументи (параметри) argc та argv наступним чином:

```
int main(int argc, char** argv)
```

або використати еквівалентний опис

```
int main(int argc, char* argv[])
```

де `argc` – довжина (кількість елементів) вектора argv;
`argv[]` – вектор вказівників на рядки, які є аргументами, що передаються у програму.

Приклад. Написати програму, яка приймає довільну кількість рядкових аргументів, розділених пробілами, і виводить їх на екран монітора.

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char** argv)
// або int main(int argc, char* argv[])
{
     int count;

     for (count=1; count < argc; count++)
          printf(" %s", argv[count]);
     printf("\n");
     return 0;
}
```

Найбільш характерним використанням аргументів головної функції є передавання їй параметрів у вигляді опцій команд ОС Linux, наприклад –l, --author.

Команди ОС Linux можуть мати опції двох видів, короткі опції та довгі опції:

- коротка опція має вигляд “-x”, де x – один символ, що символізує цю опцію, наприклад “-o”;
- довга опція має вигляд “--opt_name”, де opt_name – назва довгої (багатосимвольної) опції, наприклад “--output”.

Для зручної обробки списку аргументів програми, які є такими опціями, у стандартній бібліотеці glibc (GNU libc) існує спеціальна функція:

```
getopt_long(int argc, char* const* argv, const char* shortopts, const struct option* longopts, int* indexptr)
```

Вона декодує опції із вектору argv, повна довжина якого знаходиться у параметрі argc.

shortopts – описує список коротких опцій, які дозволені у програмі. Цей параметр є рядком символів, кожен з яких представляє собою ім'я короткої опції.

longopts – описує список довгих опцій, які дозволені у програмі.

Коли функція getopt_long знаходить коротку опцію, вона повертає символьний код короткої опції і, якщо у неї є аргумент, то заносить його у спеціальну змінну optarg.

Коли функція getopt_long знаходить довгу опцію, вона виконує дію базуючись на полях flag та val, що описані для цієї опції в структурі типу option.

Структура типу option має наступні поля:

1) const char* name – поле представляє ім'я опції;

2) int has_arg – поле вказує, чи очікує ця опція на аргументи і може приймати значення:

- no_argument
- requied_argument
- optional_argument

3) int* flag

4) int val

Два останніх поля flag та val вказують, як реагувати на появу даної опції:

- якщо вказівник flag має нульове значення, тоді функція повертає значення val;
- якщо flag є вказівником на змінну, то значення val присвоюється змінній, на яку вказує flag.

Розглянемо приклад програми, яка має один аргумент і якій у якості фактичного параметра може передаватись або коротка опція -t, або довга опція –test.

```c
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>

int main(int argc, char** argv)
{
	int c = 0;
	static int verbose_flag = 0;

	while(1)
	{

		static struct option long_options[] =
		{{"test", required_argument, 0, 't'},
		 {0, 0, 0, 0}};

		int option_index = 0;

		c = getopt_long(argc, argv, "t:", long_options,&option_index);

		if (c == -1)
			break;

		switch (c)
		{
		case 't':
			printf ("option -t /--testwith value `%s'\n", optarg);
			break;
		default:
			abort ();
		}
	}

	return 0;
}
```

Техніка налагодження програм в ОС Linux
за допомогою операторів друку printf, snprintf, fprintf

При розробці системних програм іноді виникають ситуації, в яких не доцільно користуватися програмами для налагодження типу ddd. Наприклад, це може бути у ситуації, коли виконання програми якимось чином залежить від плину часу, і призупинення програми налагоджувачем призведе до спотворення результатів, неправильної подальшої поведінки програми, тощо. У таких ситуаціях має сенс використання функцій для форматованого виводу інформації:

printf – форматований на екран;
snprintf – форматований у масив символів;
fprintf – форматований у файл.

Функція printf повинна бути вже відома студентам. Розглянемо дві інші функції.

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

Функція fprintf має із функцією printf однакову структуру рядку форматів template і відрізняється тільки тим, що виконує вивід до файлу stream.

```
int snprintf(char* s, size_t size, const char* template, ...);
```

Функція snprintf також має із функцією printf однакову структуру рядку форматів template і відрізняється тільки тим, що виконує вивід у рядок s. Крім того, ця функція перевіряє, щоб створений із template рядок не був більше розміру size (зазвичай size дорівнює розміру рядку s) і не викликає помилки сегментування пам'яті. Крім функції snprintf також існує її різновид – функція sprintf (у імені функції відсутня буква n після букви s), яка не виконує такої перевірки, і тому її використання не рекомендується.

«Найдешевшою» за часом виконання є функція snprintf тому, що вона виконує запис інформації у рядок, тобто у оперативну пам'ять. Техніка використання snprintf полягає у наступному:

1. Перед запуском основного алгоритму програми виділити пам'ять під вектор рядків фіксованої довжини;
2. Під час роботи алгоритму послідовно заносити у вектор повідомлення за допомогою функції snprintf;
3. Після закінчення роботи алгоритму виконати вивід інформації з цього вектора на екран або у файл за допомогою функцій printf, fprintf.

Додаткові можливості утиліти make

Для виклику вбудованих функцій утиліти make необхідно використовувати наступний синтаксис:

$(function arguments)

Найбільш використовувані функції:

1. shell – виконує команду із параметрами.

Наприклад “contents:=$(shell cat foo)” присвоює змінній contents вміст файлу foo.

2. wildcard – виконує пошук файлів у поточній директорії за шаблоном.

Наприклад “objects:=$(wildcard *.o)” присвоює список всіх об'єктних файлів поточної директорії змінній objects.

3. patsubst – виконує заміну у рядку згідно патерну.

Наприклад “objects:=$(patsubst %.c,%.o, main.c debug.c)” присвоює змінній objects рядок “main.o debug.o”

Також можна використовувати шаблонні правила. Наприклад, якщо Makefile містить два наступних рядки

```
%.o: %.c
        $(CC) $(CFLAGS) -c -o $@ $<
```

то рядок

%.o: %.c

вказує, що для того, щоб із довільного файлу вихідного коду (%.c) отримати відповідний об’єктний файл (%.o), необхідно виконати наступну команду, тобто

```
$(CC) $(CFLAGS) -c -o $@ $<
```

де

$(CC) – ім'я компілятора мови C, задане змінною оточення CC;

$(CFLAGS) – опції компіляції, задане змінною оточення CFLAGS;

$@ – відповідає довільному файлу результату;

$< – відповідає довільному вихідному файлу.

Створення найпростішої програми на мові Java в ОС Linux та Windows

Для створення найпростішої програми на мові **Java** необхідно створити файл з розширенням **.java** (наприклад *“Main.java”*) та внести в нього наступні рядки:

```
class Main
{
    static public void main(String[] argv)
    {
          System.out.println("Hello, world!");
    }
}
```

Для компіляції створеної програми необхідно виконати команду запуску Java- компілятора (javac):

$ javac main.java

Основні опції компілятора javac:

-g — генерувати інформацію, що використовується для налагодження програми;
-J-Xms48m — встановлює розмір початкової пам'яті у 48 Мб.

Після виконання вищенаведеної команди буде створений файл із ім'ям “main.class”. Цей файл є Java-програмою, що може виконуватись на віртуальній Java-машині. Для виконання цієї програми необхідно у директорії, де знаходиться файл-клас main.class, виконати наступну команду:

$ java main

Цією командою ми вказуємо, що необхідно викликати метод main із класу main. Метод main повинен бути оголошений із кваліфікаторами public та static, повинен не повертати жодного значення, і повинен приймати один параметр типу String[].

Створення бібліотеки (модуля) на мові Java

Типова бібліотека (модуль) на мові Java має наступний вигляд:

```
package pretty;

public class PrettyPrinter
{
    public static void prettyPrint(String[] params)
    {

     if(params.length > 0){
          System.out.println("Program argument list:");
     }

     for(int i = 0; i < params.length; i++) {
          System.out.println("Parameter #" + i + " = " + params[i]);
     }
    }
}
```

Вихідний код необхідно зберегти у файлі “PrettyPrinter.java”, а для компіляції необхідно виконати наступну команду:

$ javac -d . PrettyPrinter.java

В результаті буде створено директорію “pretty”, у якій буде збережено файл “PrettyPrinter.class”.

Тепер цю бібліотеку можна використовувати в інших Java-програмах, для чого необхідно імпортувати клас PrettyPrinter до цієї програми наступним чином:

import pretty.PrettyPrinter;

Компіляція Java програми, яка складається із декількох файлів

Основним форматом для розповсюдження Java-програм є jar-архів (Java ARchive).

Для створення jar-архіву програми, яка складається із файлу головного класу “main.class” та допоміжної бібліотеки, що знаходиться у файлі “pretty/PrettyPrinter.class”, необхідно виконати наступну команду:

```
$ jar -c main.class pretty/PrettyPrinter.class -f main.jar -m main.manifest
```

Використані опції:
-c – вказує на те, що створюється новий архів;
-f – вказує ім'я нового архіву;
-m – вказує ім'я файлу маніфесту, який буде додано до архіву.

Файл маніфесту використовується для оголошення головного класу програми. Зазвичай складається із одного рядку:

```
Main-Class: main
```

Цим вказується, що основним класом є main, у якому повинен бути оголошений статичний метод main.

Для запуску jar-архіву необхідно виконати наступну команду:

```
$ java -jar main.jar
```

Постановка завдання для програми мовою С

1. Написати програму розв’язання задачі пошуку (за варіантом) у двовимірному масиві (матриці) одним з алгоритмів методу лінійного пошуку.

2. Розміри матриці ***m*** та ***n*** взяти самостійно у межах від 7 до 10.

3. Програма **обов'язково** повинна бути написана і структурована наступним чином:

   a) оголошення структур даних (typedef) повинно бути зроблено у окремому заголовочному файлі;

   b) повинно бути щонайменше три файли із вихідним кодом (не враховуючи необхідні заголовочні файли), що міститимуть реалізації функцій введення (випадкові значення, наперед сортовані значення, з клавіатури), обробки, та виведення на друк (pretty_print) елементів матриці;

   c) для виконання завдання обробки елементів матриці повинно бути написано дві різні функції:

      1) з додатковими операторами виведення налагоджувальної інформації на друк (debug-версія);

      2) з виконанням заданих дій без додаткового виведення налагоджувальної інформації (release-версія).

4. Для компіляції написаної багатофайлової програми написати окремий make-файл, причому:

   a) при зміні одного із вихідних файлів повинен перекомпільовуватися лише цей файл (а також відбуватися дії, необхідні для генерації бінарного файлу);

   b) при видаленні бінарного файлу та незмінних вихідних файлах повинна відбуватися лише лінковка;

   c) забезпечити окрему ціль для очистки згенерованих файлів;

5. Вміти компілювати написану багатофайлову програму двома способами:

   a) за допомогою однієї команди gcc;

   b) за допомогою make-файлу.

6. Виконати тестування та налагодження програми на комп’ютері. При тестуванні програми необхідно підбирати такі вхідні набори початкових значень матриці, щоб можна було легко відстежити коректність виконання пошуку і ця коректність була б протестована для всіх можливих випадків. З метою тестування дозволяється використовувати матриці меншого розміру.

Постановка завдання для програми мовою Java

1. Написати консольну програму розв’язання задачі пошуку (за варіантом) у двовимірному масиві (матриці) одним з алгоритмів методу лінійного пошуку.
2. Розміри матриці ***m*** та ***n*** взяти самостійно у межах від 7 до 10.
3. При написанні програми повинно бути щонайменше три класи, один із яких буде відповідати за пошук елементу в матриці, другий відповідати за ввід-вивід матриці, а третій — головний клас, що міститиме метод main.
4. Для компіляції та запуску написаної програми написати окремий make-файл, причому забезпечити окремі цілі для очистки згенерованих файлів, а також генерації JAR-архіву.
5. Вміти компілювати написану програму двома способами:
    a) за допомогою однієї команди javac;
    b) за допомогою make-файлу.
6. Виконати тестування та налагодження програми на комп’ютері. При тестуванні програми необхідно підбирати такі вхідні набори початкових значень матриці, щоб можна було легко відстежити коректність виконання пошуку і ця коректність була б протестована для всіх можливих випадків. З метою тестування дозволяється використовувати матриці меншого розміру.

Зміст звіту

1. Загальна постановка задачі та завдання для конкретного варіанту.
2. Текст програми мовою C, вхідні дані.
3. Командні рядки для компілювання та запуску програми мовою C.
4. Make-файл для компілювання та запуску програми мовою C.
5. Текст програми мовою Java, вхідні дані.
6. Командні рядки для компілювання та запуску програми мовою Java.
7. Make-файл для компілювання та запуску програми мовою Java.
8. Тести для налагодження програми і результати, отримані для них на комп’ютері.

Варіанти індивідуальних завдань

Розміри матриці m і n взяти самостійно у межах від 7 до 10.

Варіант 1

Задано матрицю дійсних чисел A[m,n]. У кожному рядку матриці визначити присутність заданого дійсного числа X і його місцезнаходження (координати).

Варіант 2

Задано матрицю дійсних чисел A[n,n]. У головній діагоналі матриці знайти перший додатний і останній від’ємний елементи, а також поміняти їх місцями.

Варіант 3

Задано матрицю дійсних чисел A[m,n]. При обході матриці по рядках знайти в ній перший додатний елемент і його місцезнаходження (координати).

Варіант 4

Задано матрицю дійсних чисел A[m,n]. У кожному стовпчику матриці визначити присутність заданого дійсного числа X і його місцезнаходження (координати).

Варіант 5

Задано матрицю дійсних чисел A[n,n]. У побічній діагоналі матриці знайти перший мінімальний і останній максимальний елементи, а також поміняти їх місцями.

Варіант 6

Задано матрицю дійсних чисел A[m,n]. При обході матриці по стовпчиках знайти в ній перший мінімальний елемент і його місцезнаходження (координати).

Варіант 7

Задано матрицю дійсних чисел A[m,n]. У кожному рядку матриці знайти останній від'ємний елемент і його місцезнаходження (координати).

Варіант 8

Задано матрицю дійсних чисел A[n,n]. У головній діагоналі матриці знайти перший від'ємний і останній додатний елементи, а також поміняти їх місцями.

Варіант 9

Задано матрицю дійсних чисел A[m,n]. При обході матриці по рядках знайти в ній останній нульовий елемент і його місцезнаходження (координати).

Варіант 10

Задано матрицю дійсних чисел A[m,n]. У кожному стовпчику матриці знайти останній максимальний елемент і його місцезнаходження (координати).

Варіант 11

Задано матрицю дійсних чисел A[n,n]. У побічній діагоналі матриці знайти перший максимальний і останній мінімальний елементи, а також поміняти їх місцями.

Варіант 12

Задано матрицю дійсних чисел A[m,n]. При обході матриці по стовпчиках знайти в ній останній мінімальний елемент і його місцезнаходження (координати).

Варіант 13

Задано матрицю дійсних чисел A[m,n]. У кожному рядку матриці знайти перший максимальний елемент і його місцезнаходження (координати).

Варіант 14

Задано матрицю дійсних чисел A[n,n]. У головній діагоналі матриці знайти перший мінімальний і останній максимальний елементи, а також поміняти їх місцями.

Варіант 15

Задано матрицю дійсних чисел A[m,n]. При обході матриці по рядках визначити в ній присутність заданого дійсного числа X і його місцезнаходження (координати).

Варіант 16

Задано матрицю дійсних чисел A[m,n]. У кожному стовпчику матриці знайти перший від’ємний елемент і його місцезнаходження (координати).

Варіант 17

Задано матрицю дійсних чисел A[n,n]. У побічній діагоналі матриці знайти перший додатний і останній від’ємний елементи, а також поміняти їх місцями.

Варіант 18

Задано матрицю дійсних чисел A[m,n]. При обході матриці по стовпчиках визначити в ній присутність заданого дійсного числа X і його місцезнаходження (координати).

Варіант 19

Задано матрицю дійсних чисел A[m,n]. У кожному рядку матриці знайти перший нульовий елемент і його місцезнаходження (координати).

Варіант 20

Задано матрицю дійсних чисел A[n,n]. У головній діагоналі матриці знайти

перший максимальний і останній мінімальний елементи, а також поміняти їх місцями.

Варіант 21

Задано матрицю дійсних чисел A[m,n]. При обході матриці по рядках знайти в ній останній максимальний елемент і його місцезнаходження (координати).

Варіант 22

Задано матрицю дійсних чисел A[m,n]. У кожному стовпчику матриці знайти останній додатний елемент і його місцезнаходження (координати).

Варіант 23

Задано матрицю дійсних чисел A[n,n]. У побічній діагоналі матриці знайти перший від'ємний і останній додатний елементи, а також поміняти їх місцями.

Варіант 24

Задано матрицю дійсних чисел A[m,n]. При обході матриці по стовпчиках знайти в ній перший від'ємний елемент і його місцезнаходження (координати).

Варіант 25

Задано матрицю дійсних чисел A[m,n]. У кожному рядку матриці знайти останній мінімальний елемент і його місцезнаходження (координати).

Варіант 26

Задано матрицю дійсних чисел A[m,n]. При обході матриці по стовпчиках знайти в ній останній додатний елемент і його місцезнаходження (координати).

Варіант 27

Задано матрицю дійсних чисел A[m,n]. При обході матриці по рядках знайти в ній перший максимальний елемент і його місцезнаходження (координати).

Варіант 28

Задано матрицю дійсних чисел A[m,n]. У кожному стовпчику матриці знайти перший мінімальний елемент і його місцезнаходження (координати).

Варіант 29

Задано матрицю дійсних чисел A[n,n]. У побічній діагоналі матриці визначити присутність заданого дійсного числа X і його місцезнаходження (координати).

Варіант 30

Задано матрицю дійсних чисел A[m,n]. При обході матриці по рядках знайти в ній останній мінімальний елемент і його місцезнаходження (координати).