# Лабораторна робота №12

***Тема:*** Компілятор gcc та розробка багатофайлових програм з використанням утиліти make.

***Мета роботи:*** навчитися використовувати компілятор gcc і утиліту make для створення програм в UNIX-подібних ОС.

## *Теоретичні відомості*

Як відомо, ядро операційної системи – це програма, яка виконує основні функції операційної системи. Воно взаємодіє з апаратними пристроями, виділяє пам'ять і інші ресурси, дозволяє декільком програмам працювати одночасно, управляє файловими системами і т.д.

Ядро саме по собі не має в своєму розпорядженні засобів взаємодії з користувачами. Воно не може навіть видати простий рядок з запрошенням на введення команд. Ядро не дозволяє користувачам редагувати файли, взаємодіяти з іншим комп'ютером або писати програми. Для вирішення всіх цих задач потрібне велике число інших програм, включаючи интерпретатори команд, редактори і компілятори. Багато хто з цих програм користується бібліотеками функцій загального призначення, не включеними в ядро.

У системах GNU/Linux більшість таких програм розроблена в рамках проекту GNU (GNU – це рекурсивний акроним, який розшифровується як GNU's Not Linux (GNU – не це Linux)). Багато хто з них був написаний раніше, ніж з'явилося ядро Linux. Мета проекту GNU – «створення повноцінної операційної системи на зразок Linux, оснащеної безкоштовним програмним забезпеченням».

Ядро Linux і GNU-програми складають дуже могутню комбінацію, яку найчастіше називають просто «Linux». Але без GNU – програм система не працюватиме, як і без ядра. Тому у багатьох випадках ми говоримо GNU/Linux.

### Компілятор GCC

*Компілятор* перетворює вихідний текст програми, зрозумілий людині, в об'єктний код, що виконується комп'ютером. Компілятори, доступні в Linux-системах, являються частиною колекції GNU-компіляторів, відомої як GCC (GNU Compiler Collection). У неї входять компілятори мов C, C++, Java, Objective-C, Fortran і Chill. Нас цікавитиме компілятор з мови C.

*Компіляція вихідного файлу.*

Компілятор мови C називається *gcc*. При компіляції вихідного файлу потрібно вказувати опцію *–c.*

От як, наприклад, в режимі командного рядка компілюється файл *prog.c***:**

*$gcc –c prog.c*

Одержаний об'єктний файл називатиметься *prog.o.*

*Компоновка об'єктних файлів*

Для отримання виконуваного файлу потрібно викликати *gcc* з опцією *–o.*
*$gcc prog.c –o hello*

**Створення простої програми prog.c:**

```
#include<stdio.h>
main()
{
 printf(«Hello, World!»);
}
```

Створили текст програми і зберегли її під ім'ям *prog.c.*

Відкомпілювали програму, використовуючи компілятор *gcc.*
*$ gcc –c prog.c*
Одержимо виконуваний файл з ім'ям *hello*, скориставшись опцією *–o.*
*$ gcc prog.c –o hello*

Перевіримо працездатність і правильність виконання програми:
*$ ./hello*
В Результати успішної роботи програми на екрані повинне з'явитися вітання *Hello, world!*

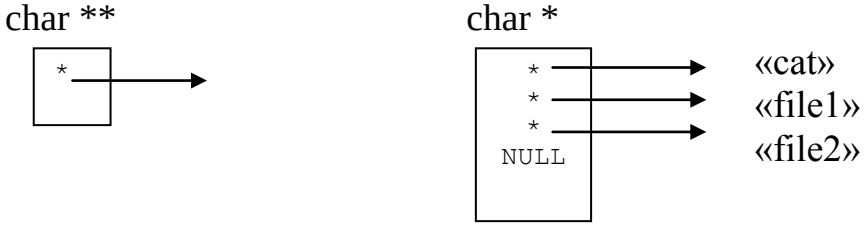
**Базова обробка командного рядка**

Програма на С дістає доступ до своїх аргументів через параметри *argc* і *argv.* Аргументи командного рядка - це інформація, слідуюча за ім'ям програми в командному рядку операційної системи.

Є два звичайні засоби визначення функції *main()* - головної функції програми на мові С:
*int main(int argc, char *argv[])*
і
*int main(int argc, char **argv)*

- Параметр *argc* є цілим числом, вказуючим кількість наявних аргументів, включаючи ім'я команди.
- Параметр *argv* - це покажчик на масив символьних покажчиків (інакше можна сказати так: *argv* є масивом покажчиків на символи).

Між двома цими оголошеннями немає різниці, хоча перше концептуально зрозуміліше, а друге технічно коректніше:




Тут *cat, file1, file2* – рядки мови С, що завершуються символом кінця рядка ‘\0’.

За угодою, *argv[0] (*у С індекси відлічуються з нуля) є ім'ям програми. Подальші елементи являються аргументами командного рядка. Останнім елементом масиву є покажчик *NULL.* Всі аргументи командного рядка - це рядки. Всі числа конвертуються програмою у внутрішній формат. Аргументи командного рядка повинні відокремлюватися пропусками. Коми, точки з комами і їм подібні символи не розглядаються як роздільники. Якщо необхідно передати рядок, що містить пропуски або табуляції, у вигляді одного аргументу, слід укласти її в подвійні лапки.

**Приклад:** Програма виводить *Hello*, а потім ім'я користувача, якщо його набрати прямо за ім'ям програми:

```
#include<stdio.h>
int main(int argc, char *argv[])
{
  if (argc!=2)
    {
     printf(«You forgot to type your name!\n»);
     return 1;
    }
  printf(«Hello %s!», argv[1]);
return 0;
}
```

Якщо назвати дану програму *name*, а як ім'я вказати *Sergey*, то в Результати виконання програми на екрані буде

*Hello, Sergey!*

У випадку, якщо програма буде запущена без вказівки аргументу, на екран буде виведено повідомлення

You forgot to type your name!

**Завдання 1**

Написати програму, яка:

- Виводить на екран ім'я програми;
- Перевіряє, чи є хоч би один аргумент і якщо так, то скільки і які.

**Завдання 2**

Написати програму, в якій перевіряється можливість відкриття на читання файлу(написаного на мові С).

**Рекомендації до виконання:**

Перед роботою з файлом його потрібно відкрити. Це можна зробити за допомогою функції *fopen():*

*FILE *fopen(const char filename, const char mode);*

Функція *fopen()* відкриває існуючий файл або створює новий.

У разі успіху вона повертає покажчик потоку, інакше повертає NULL.

Параметри функції:

*const char filename* – покажчик на рядок імені файлу. Може містити в собі інформацію про повний шлях до файлу*;*

*const char mode* – покажчик на один з можливих режимів роботи з файлом. Ось деякі з них:

*r* – читання;

*w* – запис;

*a* – додавання.

**Завдання 3:** Модифікувати програму з завдання 1 так, щоб для виведення інформації на екран використовувався стандартний потік виведення *stdout.*

**Рекомендації до виконання:**

Для роботи з файлами можна використовувати функцію *fprintf()*, яка працює практично як *printf()*, але їй потрібен аргумент для посилання на файл.

Наприклад*:*

```
int a=87;
fprintf(fp,”a=%d”,a);
```

де *fp - покажчик на файл.*

**Завдання 4**

Написати програму, яка в інтерактивному режимі читає число аргументів і самі аргументи (рядки) і виводить їх довжину.

У цьому завданні краще працювати з стандартними потоками stdin *і stdout.*

Для прочитування можна використовувати функції *fscanf()* і *fprintf(), fgets().*

Функція *fscanf()* працює аналогічно *fprintf():* прочитує дані з файлу і розносить їх по рядку, формату. Наприклад:

```
fscanf(fp, «%d»,&num);
```

Тут з файлу, на який указує файлова змінна *fp*, прочитується ціле число і поміщається за адресою змінної *num.*

Для роботи з рядками потрібно підключати файл заголовка string.h.

Функція *fgets()* зручна при порядковому прочитуванні рядків з файлу:

```
fgets(a,b,c);
```

де

*a* - рядок(покажчик), в який буде записана інформація;

b – кількість символів, які прочитуватимуться з рядка;

c – покажчик потоку файлів, тобто звідки прочитуємо.

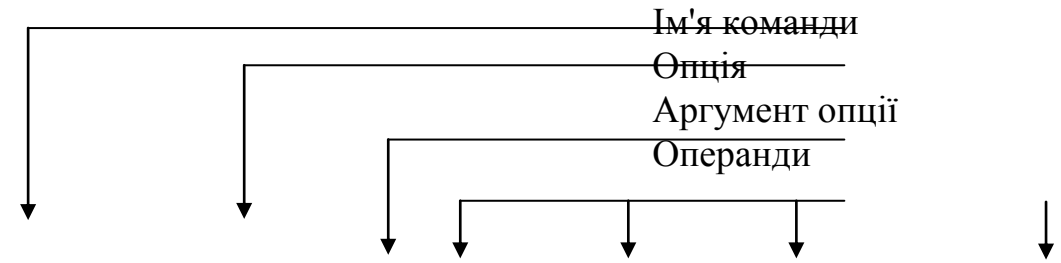
**Метод розбору командного рядка за допомогою функції *getopt().***

Розглянемо правила формування і засоби розбору командних рядків.

У загальному випадку командний рядок складається з:

- імені службової програми (утиліти);
- опцій;
- аргументів опцій;
- операндів команди.

Наприклад, в команді *grep –f patfile foo.c bar.c baz*




Розробники службових програм повинні керуватися наступними правилами.

1. Ім'я утиліти складається не менше ніж з двох і не більше ніж з дев'яти малих латинських букв і/або цифр.
2. Ім'я опції – це один буквено-цифровий символ. Опціям передує знак мінус. Після одного мінуса можуть розташовуватися декілька опцій без аргументів.
3. Опції відокремлені від своїх аргументів.
4. У опцій немає необов'язкових аргументів.
5. Якщо у опції декілька аргументів, вони представляються одним словом і відокремлюватися один від одного комами або екранованими пропусками.
6. Всі опції розташовуються в командному рядку перед операндами.
7. Спеціальний елемент командного рядка --, який не є ні опцією, ні операндом, позначає кінець опцій. Всі подальші слова трактуються як операнди, навіть якщо вони починаються із знаку мінус.
8. Порядок різних опцій в командному рядку не має значення. Якщо повторюється одна опція з аргументами, останні повинні інтерпретуватися в порядку, вказаному в командному рядку.
9. Порядок інтерпретації операндів може залежати від утиліти.
10. Якщо операнд задає читаний або записуваний файл, то знак мінус на його місці використовується тільки для позначення стандартного введення (або стандартного виведення , якщо з контексту ясно, що специфікується вихідний файл).

Для розбору опцій і їх аргументів засобами мови С служить функція *getopt()* і асоційовані з нею зовнішні змінні.

```
#include<unistd.h>
int getopt(int argc,char *const argv[],const char *optstring);
extern char *optarg;
extern int optind,opterr,optopt;
```

Тут:

- аргументи *argc* і *argv* задають командний рядок в тому вигляді, як він передається функції *main();*
- *optstring* є ланцюжком імен опцій;
- змінна *optind* (із початковим значенням 1) служить індексом в масиві *argv[]*.

У Результати функція *getopt()* повертає ім'я чергової опції з числа перерахованих в ланцюжку *optstring(*якщо таке вдалося виділити*).* За наявності у опції аргументу покажчик на нього поміщається в змінну *optarg* з відповідним збільшенням значення *optind.*

Якщо у опції аргумент відсутній, а на першому місці в *optstring* задано двокрапку, воно і служить результатом.

Якщо зустрілося ім'я опції, не перераховане *optstring,* або у опції немає аргументу, а на першому місці в *optstring* задано не двокрапку, то результатом стане знак питання.

У будь-якій з перерахованих вище помилкових ситуацій в змінну *optopt* поміщається ім'я «проблемної» опції. Крім того, в стандартний протокол видається діагностичне повідомлення. Для придушення видачі слід привласнити змінній *opterr* нульове значення.

Нарешті, якщо при виклику *getopt()* покажчик *argv[optind]* не відзначає початок опції (наприклад, він пустий або перший символ вказуємого ланцюжка відмінний від знаку мінус), результат рівний –1 як ознака того, що розбір опцій закінчений.

**Завдання**

Написати програму обробки командного рядка виклику програми, що приймає опцію *–a* без аргументу і опцію *–b* з аргументом.

## Розробка багатофайлових програм. Використання утиліти make

*Компіляція програм, що містять дві і більш функцій*

Як відомо, з метою підвищення рівня структурованості, програма на мові С є набором функцій. Кожна функція С в програмі має рівні права з іншими функціями. Кожна з них може викликати будь-яку іншу функцію або викликатися будь-якою іншою функцією.

Чи не є функція main() особливою? Так, вона дещо відрізняється від інших тим, що, коли програма, що складається з декількох функцій, збирається воєдино, виконання починається з першого оператора у функції main(); але ця єдина перевага, що надається їй. Навіть функція main() може викликатися рекурсивно або іншими функціями, хоча на практиці це робиться рідко.

Простий підхід до використання декількох функцій - приміщення їх в один файл. Потім досить скомпілювати цей файл, неначебто він містив єдину функцію. Але такий підхід прийнятний тільки для простих, учбових програм. На практиці при розробці реальних проектів призначають різним завданням різні функції, що сприяє поліпшенню програми.

Припустимо, що в системі Linux встановлений компілятор GNU C - gcc і file1.c, file2.c - два файли, що містять функції С. Тоді наступний командний

рядок скомпілює обидва файли і створить виконуваний файл, ім'я за умовчанням якого a.out.

**$gcc file1.c file2.c**

Крім того, при цьому створюються два об'єктні файли, названі file1.o file2.o. Якщо згодом змінити файл file1.c, але залишити file2.c незмінним, можна скомпілювати перший файл і об'єднати його з об'єктною версією другого файлу, скориставшись командою:

**$gcc file1.c file2.o**

**Використання файлів заголовків**

Якщо помістити функцію main(), а визначення функцій - в другий, перший файл все ж таки потребуватиме прототипів функцій. Замість того, щоб вводити їх при кожному використанні файлу функції, прототипи функцій можна зберігати у файлі заголовка. Файли заголовків містять визначення функцій - число передаваних аргументів, типи аргументів і повертаного значення. Саме це і робить стандартна бібліотека С, наприклад, поміщаючи прототипи функцій введення/виведення у файл stdio.h. Більшість системних файлів заголовків розташована в каталогах /usr/include або /usr/include/sys.

*Автоматизація процесу створення програми за допомогою GNU-утиліти make*

Утиліта make дозволяє автоматично перекомпілювати програму. Основна ідея утиліти make проста. Їй указуються *цільові модулі*, що беруть участь в процесі побудови виконуваного файлу, і *правила,* по яких протікає цей процес. Також задаються *залежності,* що визначають, коли конкретний цільовий модуль повинен бути перебудований. Крім очевидних цільових модулів повинен також існувати модуль *clean.* Він призначений для видалення всіх об'єктних файлів, що згенерували, і програм, щоб можна було почати все з початку. Правило для даного модуля включає команду rm, що видаляє перераховані файли:

clean:

rm –f *.o <ім’я виконувального файла>

Щоб передати всю інформацію про процес створення програми утиліті make, необхідно створити файл Makefile. У ньому цільові модулі перераховуються зліва. За ім'ям модуля слідує двокрапка і існуючі залежності. У наступному рядку указується правило, по якому створюється модуль. Рядок правила повинен починатися з символу табуляції, інакше утиліта make проінтерпретує її неправильно.

**Завдання:**

- Написати багатофайлову С-программу;

- Виконати компіляцію і компоновку за допомогою строкового компілятора GNU C;
- Автоматизувати процес створення програми за допомогою GNU-утиліти make.

**Завдання**

Номер варіанту визначається за номером запису призвища в журналі для проведення лабораторних робіт.

1. Написати програму, яка забезпечує роботу команди вигляду: [ім’я_команди] [-a|-b] [-o вихідний файл], де -a і -b несумісні прапори.

- якщо заданий прапор -a команда відображає час в секундах, що пройшов з моменту завантаження системи;
- якщо заданий прапор -b команда відображає число процесів, що працюють в системі;

Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

2. Написати програму, яка забезпечує роботу команди вигляду:
[им’я_команди] [-a|-b] [-o вихідний файл],
де -a і -b несумісні прапори.

- якщо заданий прапор -a - команда відображає ім'я операційної системи, номер версії і модифікації ядра;
- якщо заданий прапор -b – інформацію про платформу, на якій працює система.

Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

3. Написати програму, яка забезпечує роботу команди вигляду:
[ім’я_команди] [-a|-b] [-o вихідний файл],
де -a і -b несумісні прапори.

- якщо заданий прапор -a - команда відображає назва домашньої , в якому користувач входу в систему;
- якщо заданий прапор -b – ім'я терміналу.

Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

4. Написати програму, яка забезпечує роботу команди вигляду: [ім’я_команди] [-a|-b] [-o вихідний файл], де -a і -b несумісні прапори.

- якщо заданий прапор -a - команда відображає пошуковий шлях;
- якщо заданий прапор -b – зареєстроване ім'я користувача.

Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

5. Написати програму, яка забезпечує роботу команди вигляду: [ім’я_команди] [-a|-b] [-o вихідний файл], де -a і -b несумісні прапори.

- якщо заданий прапор -a - команда відображає використовуване віконне середовище;

- якщо заданий прапор -b – ім'я хоста.

Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

6. Написати програму, яка забезпечує роботу команди вигляду:
[им’я_команди] [-a|-b] [-o вихідний файл], де -a і -b несумісні прапори.

- якщо заданий прапор -a - команда відображає ім'я поштової скриньки;
- якщо заданий прапор -b – ім'я поточного каталога.

Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

7. Написати програму, яка забезпечує роботу команди вигляду:
[ім’я_команди] [-a|-b] [-o вихідний файл], де -a і -b несумісні прапори.

- якщо заданий прапор -a - команда відображає шлях до тимчасового каталога користувача;
- якщо заданий прапор -b – використовувану SHELL-оболонку. Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

8. Написати програму, яка забезпечує роботу команди вигляду: [ім’я_команди] [-a|-b] [-o вихідний файл], де -a і -b несумісні прапори.

- якщо заданий прапор -a - команда відображає час в секундах, що пройшов з моменту завантаження системи;
- якщо заданий прапор -b – число процесів, що працюють в системі.

Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

9. Написати програму, яка забезпечує роботу команди вигляду: [ім’я_команди] [-a|-b] [-o вихідний файл], де -a і -b несумісні прапори.

- якщо заданий прапор -a - команда відображає загальний об'єм оперативної пам'яті;
- якщо заданий прапор -b – вільний об'єм ОЗУ.

Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

10. Написати програму, яка забезпечує роботу команди вигляду:
[ім’я_команди] [-a|-b] [-o вихідний файл],
де -a і -b несумісні прапори.

- якщо заданий прапор -a - команда відображає ім'я операційної системи, номер версії і модифікації ядра;
- якщо заданий прапор -b – ім'я домашнього директория, в якому користувач виявляється після входу в систему.

Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

11. Написати програму, яка забезпечує роботу команди вигляду:
[ім’я_команди] [-a|-b] [-o вихідний файл], де -a і -b несумісні прапори.

- якщо заданий прапор -a - команда відображає інформацію про платформу, на якій працює система;

- якщо заданий прапор -b – ім'я терміналу. Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

12. Написати програму, яка забезпечує роботу команди вигляду: [ім'я_команди] [-a|-b] [-o вихідний файл], де -a і -b несумісні прапори.

- якщо заданий прапор -a - команда відображає інформацію про платформу, на якій працює система;
- якщо заданий прапор -b – використовувану SHELL-оболонку

Одержана інформація також записується у файл, ім'я якого задається як аргумент опції -o.

**Вимоги до оформлення завдання**

Результати виконання завдання надаються повинен містити всі тексти програми, а також всю решту файлів, необхідних для компіляції і запуску виконаної роботи.

Опис розрахунково-графічного завдання повинен містити:

1. постановку завдання;
2. опис всіх модулів, що містять текст програми;
3. опис всіх використаних змінних;
4. інструкцію з експлуатації програми;
5. контрольні завдання і відповідні результати (тести);

**Контрольні питання:**

1. Що є багатофайловою програмою на мові С?
2. Як створюються багатофайлові програми?
3. Як і з якою метою в багатофайлових програмах використовують файли заголовків для функцій, що викликаються?
4. Для чого призначена GNU-утиліта make?
5. Що є Makefile і які вимоги по його складанню?
6. Чим відрізняються системнівиклики від функцій бібліотеки загального призначення?
7. Поясніть призначення і наведіть формат системного виклику open()?
8. Як здійснити читання і записування файлів за допомогою системних викликів?
9. Яке призначення системного виклику lseek()?
10. Як отримати інформацію про атрибути файла?
11. Як виконати читання вмісту каталогу за допомогою системних викликів?
12. Як традиційно оголошується функція main() згідно стандарту ANSI? Згідно стандарту POSIX?