

```
int fct( int n){  
    return n<2?1:fct(n-1);  
}
```

## Лекція fct(3)

Робота з багатофайловими застосуваннями

**// файл 1 add.c**

```
int add(int x, int y){  
    return x+y;  
}
```

**// файл 2: main.c**

```
#include <stdio.h>  
  
int main(){  
    printf("The sum of 3 and 4 is: %d \n ", add(3, 4));  
}
```

**// gcc main.c**

**//warning: implicit declaration of function 'add' [-Wimplicit-function-declaration]**



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

```
#include "add.c"
```

```
int main(){
```

```
    printf("The sum of 3 and 4 is: %d \n ", add(3, 4));
```

```
}
```

**//gcc main.c – OK**

/\* #include "add.c" просто додає код файлу add.c до другого файлу main.c, а отже команда компіляції ( наприклад gcc main.c ) обробить два файли як один зконкатенований

```
int add(int x, int y){
```

```
    return x+y;
```

```
}
```

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

```
int main(){
```

```
    printf("The sum of 3 and 4 is: %d \n ", add(3, 4));
```

```
}
```



**# include <filename.h>** - шукає файли в спеціальній директорії, призначеній для стандартних заголовочних файлів. Наприклад, `stdio.h` , `math.h` та інші

**# include "filename.h"** - шукає файл в директорії для користувацьких заголовочних файлів (зокрема, у тій самій директорії, що й текст модуля)

**//Файл main.c** (з попередньою декларацією - forward declaration):

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

```
extern int add(int x, int y); // декларація функції, вказуємо що є функція add()
```

```
// без специфікатору extern в принципі можна обійтися
```

```
int main(){
```

```
    printf("The sum of 3 and 4 is: %d \n ", add(3, 4));
```

```
    return 0;
```

```
}
```

**// Файл add.cpp :**

```
int add(int x, int y){
```

```
    return x+y;
```

```
}
```

```
>>> gcc main.c add.c
```

**//Файл add.c :**

```
int add(int x, int y){  
    return x + y;  
}
```

**// Файл add.h :**

```
extern int add(int x, int y);
```

**// Файл main.c:**

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

```
#include "add.h"
```

```
int main(){  
    printf("The sum of 3 and 4 is: %d \n ", add(3, 4));  
    return 0;  
}
```

**>>> gcc main.c add.c**

### **Source1.h:**

```
typedef int ZINT;  
ZINT func();
```

### **Source1.c**

```
ZINT func(){  
    return 0;  
}
```

### **Source2.h :**

```
#include "Source1.h"  
ZINT func2();
```

### **Source2.c**

```
ZINT func2(ZINT x){  
    return z+x;  
}
```

**Main.c :** Source1.h Source2.h -- проблема подвійного включення

```
# ifndef _RATIO_H_
# define _RATIO_H_
typedef struct tagRatio {
    int m , n ;
} TRatio ;
```

```
extern int inputRatio(TRatio* z);
extern void printRatio(TRatio z);
extern TRatio addRatio(TRatio a, TRatio a);
extern TRatio subRatio(TRatio a, TRatio a);
****
```

```
# endif /* end of _RATIO_H_ */
```

```
#ifndef _FILENAME_H
#define _FILENAME_H
```

```
// Types ...
// Functions declarations...
```

```
#endif /* end of _FILENAME_H */
```



```
/* Файл- file.h: */
```

```
#ifndef __cplusplus
```

```
extern "C" {
```

```
#endif
```

```
// Header declarations
```

```
#ifndef __cplusplus
```

```
} // end extern "C"
```

```
#endif
```

```
/* Файл- ratio.h: */
```

```
#ifndef _RATIO_H_
```

```
#define _RATIO_H_
```

```
#ifndef __cplusplus
```

```
extern "C" {
```

```
#endif
```

```
typedef struct tagRatio {
```

```
int m , n ;
```

```
} TRatio ;
```

```
extern TRatio mul(TRatio a, TRatio b);
```

```
#ifndef __cplusplus
```

```
} // end extern "C"
```

```
#endif
```

```
#endif /* end of _RATIO_H_ */
```

**insert.h** -> Містить декларацію стр-ри Node, декларацію функції insertion.

**linkedlist.h** -> Стр-ра Node та декларацію функції Display яку потрібно всюди підключати.

**insert.c** -> Включає декларацію Node через #include "linkedlist.h" та містить реалізацію функції з insert.h.

**linkedlist.c** -> Огортку(Wrapper) для користування функціями на зразок Insert та відображення списку

## // linkedlist.h

```
#ifndef LINKED_LIST_H
#define LINKED_LIST_H

struct Node {
    int data;
    struct Node* next;
};

void display(struct Node* temp);

#endif
```

## // insert.h

```
#ifndef INSERT_H
#define INSERT_H

struct Node;
struct Node* create_node(int data);
void b_insert(struct Node** head, int data);
void n_insert(struct Node** head, int data, int pos);
void e_insert(struct Node** head, int data);

#endif
```

## // insert.c

```
#include "linkedlist.h"
```

```
// to tell the preprocessor to look into the current directory and standard library files later.
```

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

```
struct Node* create_node(int data) {  
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));  
    temp->data = data;  
    temp->next = NULL;  
    return temp;  
}
```

```
void b_insert(struct Node** head, int data)  
{  
    struct Node* new_node = create_node(data);  
    new_node->next = *head;  
    *head = new_node;  
}  
****
```

## / linkedlist.c - Driver Program

```
#include "insert.h"
#include "linkedlist.h"
#include <stdio.h>
void display(struct Node* temp) {
    printf("The elements are:\n");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    struct Node* head = NULL;
    int ch, data, pos;
    while (1) {
        printf("1.Insert at Beginning"); printf("\nEnter your choice: ");
        scanf("%d", &ch);
        printf("Enter the data: ");
        scanf("%d", &data);
        b_insert(&head, data); **** }
}
```



1)

gss файл1.с файл2.с файл3.с –о проект


2)

gss -с файл1.с

gss -с файл2.с

gss -с файл3.с

gss файл1.о файл2.о файл3.о –о проект



## Бібліотеки

- Бібліотека - це набір скомпонованих особливим чином об'єктних файлів. Бібліотеки підключаються до основної програми під час компонування. За способом компонування бібліотеки поділяють на архіви (статичні бібліотеки, static libraries) і спільно використовуються (динамічні бібліотеки, shared libraries).

## Статичні бібліотеки

**Статичні бібліотеки** - це набір вже скомпільовані підпрограм або об'єктів, які підключаються до оригінального пакету у вигляді об'єктних файлів. Цей набір виглядає як архів з декількох об'єктних файлів, який вміє розпаковувати gcc.

Розширення: .a (Linux) та .lib (Win)

Статична лінковка:

```
>>>gcc -c file.c # file.o  
>>>gcc file.o -o file # file  
>>> gcc -lm file  
>>>gcc file.o -l stdc++ -o file
```



## Динамічні бібліотеки

- Динамічна бібліотека мови - це бібліотека, яка завантажується в ОС за запитом працюючої програми безпосередньо в ході її виконання. Це робить лінковщик, який збирає цю бібліотеку з програмних модулів, і завантажувач, який при запуску перевіряє наявність цих модулів на комп'ютері.
- На відміну від статичних бібліотек, код спільно використовуваних (динамічних) бібліотек не включається в бінарний файл. Замість цього в нього включається тільки посилання на бібліотеку.
- Переваги динамічної компоновки в тому, що спільні бібліотеки займають менше місця, і, якщо ми робимо якісь оновлення, то виконуваний файл перекомпілювати не потрібно
- Розширення .so в linux и .dll в win

## Опис проекту

- Каталоги library і project знаходяться в загальному каталозі User.
- Каталог library містить каталог source (c-files). Також в library будуть знаходитися заголовки (h-files), статична (libmy1.a) і динамічна (libmy2.so) бібліотеки.
- Каталог project буде містити файли вихідних кодів проекту та заголовки з описом функцій проекту. Тут буде розташовуватися виконуваний файл проекту.
- В операційних системах GNU / Linux імена файлів бібліотек повинні мати префікс "lib", статичні бібліотеки - розширення \* .a, динамічні - \* .so.

Файл figure.c:

```
void rect (char sign, int width, int height) {
    int i, j;

    for (i=0; i < width; i++) putchar(sign);
    putchar('\n');

    for (i=0; i < height-2; i++) {
        for (j=0; j < width; j++) {
            if (j==0 || j==width-1)
                putchar(sign);
            else putchar(' ');
        }
        putchar('\n');
    }

    for (i=0; i < width; i++) putchar(sign);
    putchar('\n');
}
```

```
void diagonals (char sign, int width) {
    int i, j;

    for (i=0; i < width; i++) {
        for (j=0; j < width; j++) {
            if (i == j || i+j == width-1)
                putchar(sign);
            else putchar(' ');
        }
        putchar('\n');
    }
}
```

## Файл заголовків та текст.c

```
//Файл text.c:  
void text (char *ch) {  
    while (*ch++ != '\0') putchar('*');  
    putchar('\n');  
}
```

```
//Файл mylib.h:  
void rect (char sign, int width, int  
height);  
void diagonals (char sign, int width);  
void text (char *ch);
```

## Компіляція статичної бібліотеки

```
>>cd library  
>>gcc -c ./source/*.c  
# figures.o mylib.h source text.o  
>>> ar r libmy1.a *.o  
>>>rm *.o # не обов'язково  
# libmy1.a mylib.h source
```

# Компіляція динамічної бібліотеки

```
>>>gcc -c -fPIC source/*.c
```

```
>>>gcc -shared -o libmy2.so *.o
```

```
>>>rm *.o
```

```
# libmy1.a libmy2.so mylib.h source
```

## Файли директорії project

```
//Файл data.c:
#include <stdio.h>
#include "../library/mylib.h"
void data (void) {
    char str[3][30];
    char *prompts[3] = {"Name: ", "Place: ", "Point "};
    int i;


    for (i=0; i<3; i++) {
        printf("%s", prompts[i]);
        gets(str[i]);
    }
    diagonals('~', 7);
    for (i=0; i<3; i++) {
        printf("%s", prompts[i]);
        text(str[i]);
    }
}
```

```
//Файл main.c:
#include <stdio.h>
#include
"../library/mylib.h"
#include "project.h"

main () {
    rect('-',75,4);
    data();
    rect('+',75,3);
}
```



//Файл project.h

```
void data (void);
```



```
>>>cd ../project
>>>gcc -c *.c
# main.o и data.o
>>> gcc -o project *.o -L../library -lmy1

>>>gcc -o project *.o -L../library -lmy2 -Wl,-rpath,../library/
```





## Project 2

```
/* world.h */  
void h_world (void);  
void g_world (void);
```

```
/* h_world.c */  
#include <stdio.h>  
#include "world.h"  
void h_world (void){  
    printf ("Hello World\n");  
}
```

```
/* g_world.c */  
#include <stdio.h>  
#include "world.h"  
void g_world (void)  
{  
    printf ("Goodbye World\n");  
}
```

```
/* main.c */  
#include "world.h"  
  
int main (void)  
{  
    h_world ();  
    g_world ();  
}
```

## Makefile (static)

# Makefile for World project

binary: main.o libworld.a

**gcc -o binary main.o -L. -lworld**

main.o: main.c

**gcc -c main.c**

libworld.a: h\_world.o g\_world.o

**ar cr libworld.a h\_world.o g\_world.o**

h\_world.o: h\_world.c

**gcc -c h\_world.c**

g\_world.o: g\_world.c

**gcc -c g\_world.c**

clean:

**rm -f \*.o \*.a binary**



```
>>> make
```

```
>>>gcc -c main.c
```

```
>>>gcc -c h_world.c
```

```
>>>gcc -c g_world.c
```

```
>>>ar cr libworld.a h_world.o g_world.o
```

```
>>>gcc -o binary main.o -L. -lworld
```

```
>>>./binary
```

```
Hello World
```

```
Goodbye World
```



## Makefile (dynamic)

# Makefile for World project

binary: main.o libworld.so

**gcc -o binary main.o -L. -lworld -Wl,-rpath,.**

main.o: main.c

**gcc -c main.c**

libworld.so: h\_world.o g\_world.o

**gcc -shared -o libworld.so h\_world.o g\_world.o**

h\_world.o: h\_world.c

**gcc -c -fPIC h\_world.c**

g\_world.o: g\_world.c

**gcc -c -fPIC g\_world.c**

clean:

**rm -f \*.o \*.so binary**

  
**cmake\_minimum\_required(VERSION 2.8)**

**## Use the variable PROJECT\_NAME for changing the target name**  
**set( PROJECT\_NAME "HelloWorld" )**

**## Set our project name**  
**project(\${PROJECT\_NAME})**

**## Set include folder**  
**include\_directories ("Headers")**

**## Use all the \*.cpp files we found under this folder for the project**  
**FILE(GLOB SRCS "\*.cpp" "source/\*.c")**

**## Use all the \*.h files we found under this folder for the project**  
**FILE(GLOB HDRS "\*.h" "\*.hpp")**

**## Define the executable**  
**add\_executable(\${PROJECT\_NAME} \${HDRS} \${SRCS})**



# За межами курсу

- Перерахування (enumerations, enum)
- Об'єднання (unit)
- Оператори switch, goto
- Asm (за межами курсу)
- Макроси
- Вказівники на функцію
- Робота з системою (за межами стандарту)
- Широкі символи
- Робота з потоками та перериванням