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

Лекція fct(3)

Робота з багатофайловими застосуваннями

Робота з двома С-файлами

```
// 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]
```

Робота з двома С-файлами

```
#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));
}  */
```

Макрокоманда(макрос) #**include**

#include <filename.h> - шукає файли в спеціальній директорії(-ях) (системні PATH), призначеній для стандартних заголовочних файлів. Наприклад, stdio.h , math.h та інші

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

Робота з двома С-файлами

//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;
}
```

Компіляція:

>>> gcc main.c add.c

Робота з двома С-файлами

```
//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;
}
```

Компіляція:

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

Подвійне включення

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

Макроси для безпечного 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 */
```

Макроси (#define)

```c
#include <stdio.h>
#include "my_header.h"


#include <stdio.h>
#define PI 3.1415
int main(){
        float radius, area;
        printf("Enter the radius: ");
        scanf("%f", &radius);
        area = PI*radius*radius; // PI - макрос
printf("Area=%.2f",area);
}
```

```c
#include <stdio.h>
#define PI 3.1415
#define circleArea(r) (PI*r*r)
int main() {
    float radius, area;
    printf("Enter the radius: ");
    scanf("%f", &radius);
    area = circleArea(radius); // макрос
    printf("Area = %.2f", area);
    return 0;
}
```

Макроси умов

#pragma - опція компілятора

```
#ifdef <MACRO>        // умова (чи визначений MACRO)
   // якийсь код
#endif
```

#if, #elif and #else

```
#if <вираз>    // умова якщо <вираз>  не дорівнює 0
  // якийсь код
#elif <вираз1>    // додаткова умова якщо <вираз1>  не дорівнює 0
  // інший код
#elif <вираз2>    // додаткова умова якщо <вираз2>  не дорівнює 0
   // а тут ще щось
#else    // інакше …
  // взагалі щось інше
#endif
// поїхали далі — цей код працює всюди
```

Включення C у C++

```
/* Файл-  file.h: */
#ifdef __cplusplus
extern "C" {
#endif

// Header declarations

#ifdef __cplusplus
} // end extern "C"
#endif
```

```
/* Файл-  ratio.h: */
# ifndef _RATIO_H_
# define _RATIO_H_

#ifdef __cplusplus
extern "C" {
#endif
typedef struct tagRatio {
int m , n ;
} TRatio ;

extern TRatio mul(TRatio a, TRatio b);
#ifdef __cplusplus
} // end extern "C"
#endif

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

Приклад проекту

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 реалізацю функції з insert.h.
linkedlist.c -> Огортку(Wrapper) для користуванням функціями на зразок Insert та відображення списку

Приклад проекту

// 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
```

Приклад проекту

```
// Файл 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;
}
****
```

Приклад проекту

```
// Файл 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); **** }
```

Компіляція проекту

1) **Разом об’єктні файли та весь проект**

gcc файл1.c файл2.c файл3.c –o <назва>

2) **Окремо об’єктні файли – потім проект**

gcc -c файл1.c

gcc -c файл2.c

gcc -c файл3.c

gcc файл1.o файл2.o файл3.o –o <назва>

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
ті, що спільно використовуються (***динамічні бібліотеки, shared libraries).***

Статичні бібліотеки

Статичні бібліотеки - це **набір вже скомпільованих 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 (Linux gcc):

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

Динамічні бібліотеки

• **Динамічна бібліотека** - це бібліотека, яка з**авантажується в ОС за запитом працюючої програми** безпосередньо **в ході її виконання**. Це робить лінковщік, 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 компонування в тому, що спільні бібліотеки займають **менше місця**, і, якщо ми робимо якісь оновлення, то виконуваний файл **перекомпілювати не потрібно.**

• Розширення .so в Linux(Unix) и .dll в Windows

Опис проекту

•Каталоги library і project знаходяться в загальному каталозі User.

•Каталог library містить каталог source (ісходні файли, c-files). Також в library будуть знаходитися заголовочні файли (h-files), статична (libmy1.a) і динамічна (libmy2.so) бібліотеки.
(В операційних системах GNU / Linux імена файлів бібліотек повинні мати префікс "lib", статичні бібліотеки - розширення * .a, динамічні - * .so.)

• Каталог project буде містити файли вихідних кодів проекту та заголовки з описом функцій проекту. Тут буде розташовуватися виконуваний файл проекту.

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');
}
```

Заголовочний файл mylib.h та драйвер text.c

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


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

Компіляція статичної бібліотеки

```
>>cd library
>>gcc -c ./source/*.c
# figures.o  mylib.h  source  text.o
>>> ar r libmy1.a *.o
>>>rm *.o # не обовязково
# 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 strs[3][30];
   char *prompts[3] = {"Name:”, “Place”,"Point "};
   int i;
   for (i=0; i<3; i++) {
    printf("%s", prompts[i]);
    gets(strs[i]);
  }
  diagonals('~', 7);
  for (i=0; i<3; i++) {
    printf("%s", prompts[i]);
    text(strs[i]);
  }
}
```

```
//Файл main.c:
#include <stdio.h>
#include "../library/mylib.h"
#include "project.h“
int main () {
   rect('-',75,4);
   data();
   rect('+',75,3);
}
```

```
//Файл project.h
/*Метод data */
void data (void);
```

Компіляція розом з бібліотеками

```
>>>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/
```

Файли Project 2

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

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

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

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

Makefile (static)

```
# Makefile for World project
#  Формат:
#<Ціль>: <вхідні файли>
#       <команда>
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
```

Результат роботи

```
>>> 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: CMakefile.txt

```
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)
- Обєднання (union), бітові структури (a:b)
- Комплексний тип

- Оператори switch, goto
- Asm (за межами курсу)
- Базові алгоритми, локалізація та робота з часом

- Макроси
- Вказівники на функцію
- Функції з довільною кількістю аргументів
- Широкі символи

- Робота з потоками та перериваннями,C11-C20
- Робота з системою (за межами стандарту)