

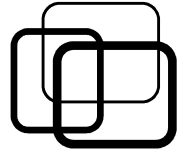
Function and struct

Inst. Nguyễn Minh Huy

Contents



- Function.
- Multiple-file project.
- struct.



■ Problem with repeated code:

■ Consider a program:

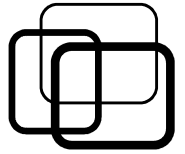
- Enter 3 positive integers $a, b, c \geq 0$.
- Compute and print $S = a! + b! + c!$.

➔ Identify repeated code.

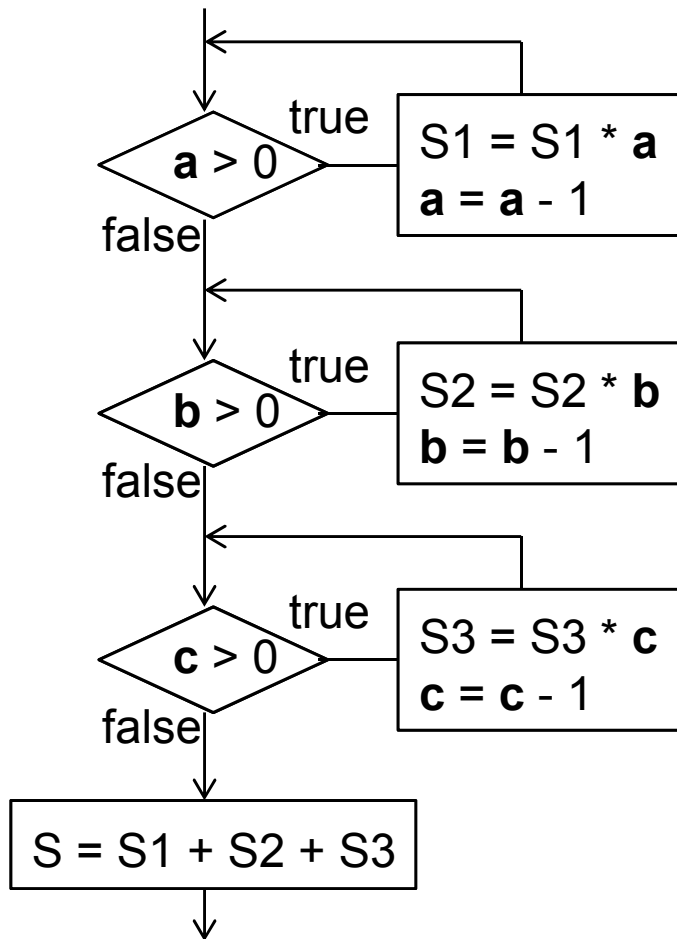
■ Disadvantages of repeated code:

- Time and cost.
- Changes → fix multiple places.
- ➔ Write once, reuse everywhere.

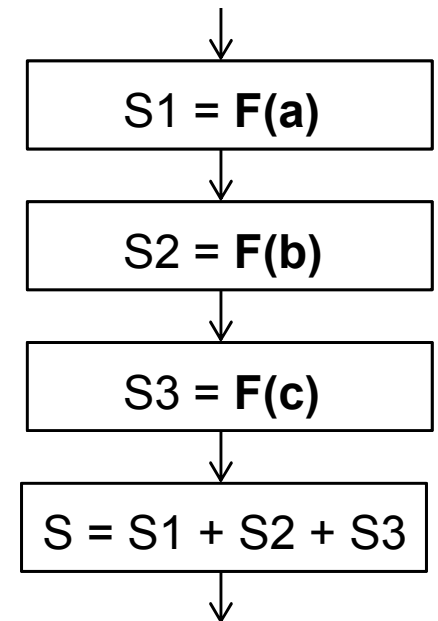
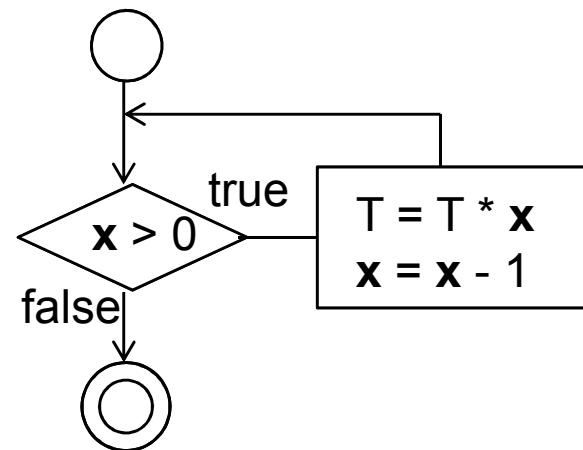
Function



■ Function solution:



Define $F(x)$



Function

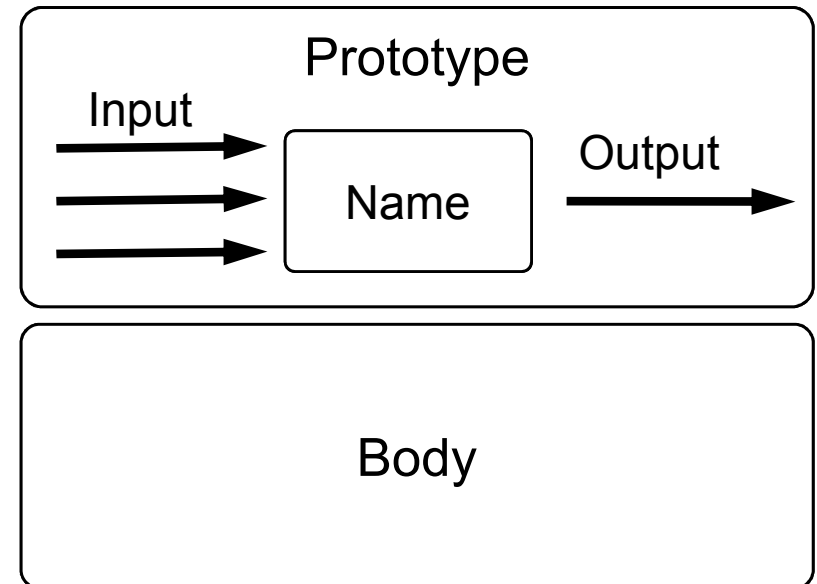


■ C/C++ function:

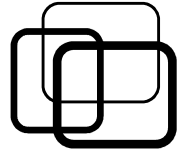
- A named block of statements.
- Can be called:
 - From anywhere in program.
 - With different arguments.

■ Function structure:

- Prototype: declaration.
 - Function name.
 - Arguments.
 - Return type.
 - ➔ Identification.
- Body: implementation.



Function



■ C/C++ function:

■ Declaration (prototype):

<Return type> **<Function name>**(*<Arguments>*);

<Return type>: int, float, char, ..., **void** (no return).

float **calcGPA**(float literature, float math);

void **printResult**();

■ Implementation (body):

<Return type> **<Function name>**(*<Arguments>*)

{

 [Statements]

 [**return** *<value>*;]

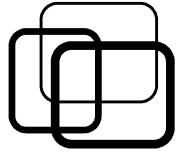
}

■ Calling:

<Function name>(*<Arguments>*);

float gpa = **calcGPA**(7.0, 8.5);

Function



■ C/C++ function:

// Function declaration.

```
long long factorial( int n );
```

```
int main()
```

```
{
```

```
    // Declare and input a, b, c.
```

```
    // Function calls.
```

```
    S1 = factorial( a );
```

```
    S2 = factorial( b );
```

```
    S3 = factorial( c );
```

```
    S = S1 + S2 + S3;
```

```
}
```

// Function implementation.

```
long long factorial( int n )
```

```
{
```

```
    long long s = 1;
```

```
    for ( ; n > 0; n--)
```

```
        s = s * n;
```

```
    return s;
```

```
}
```



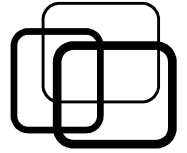
■ Passing arguments:

■ Pass-by-value:

- Argument values are passed to function.
- Function receives only the COPY.
- Real arguments are UNCHANGED.
- Arguments are: variables, constant, expressions.

```
float calcGPA( float lit, float math )  
{  
    lit = lit * 2;  
    math = math * 3;  
    return (lit + math) / 5;  
}
```

```
int main()  
{  
    float a, b, gpa;  
  
    gpa = calcGPA( a, b );  
    gpa = calcGPA( 6, 8.5 );  
    gpa = calcGPA( a + 1, b );  
    // a, b are UNCHANGED.  
}
```

■ Passing arguments:

■ Pass-by-reference (C++):

- Real arguments are passed to function.
- Function receives the original ones.
- Real arguments can be CHANGED.
- Arguments are variables only.
- Syntax: **&**<argument>.

```
float calcGPA( float &lit, float math )  
{  
    lit = lit * 2;  
    math = math * 3;  
    return (lit + math) / 5;  
}
```

```
int main()  
{  
    float a, b, gpa;  
  
    gpa = calcGPA( a, b);  
    // a is CHANGED.  
    gpa = calcGPA( 6, 8.5 );    //wrong  
    gpa = calcGPA(a + 1, b);    //wrong  
}
```

Function



■ Passing arguments:

■ Notes:

➤ Use pass-by-reference to return values.

➔ Function with multiple return values.

```
void input( float &lit, float &math )
```

```
{
```

```
    printf("Enter literature = ");
```

```
    scanf("%f", &lit);
```

```
    printf("Enter math = ");
```

```
    scanf("%f", &math);
```

```
}
```

```
void calcGPA( float lit, float math, float &gpa )
```

```
{
```

```
    lit = lit * 2;
```

```
    math = math * 3;
```

```
    gpa = (lit + math) / 5;
```

```
}
```

```
int main()
```

```
{
```

```
    float a, b;
```

```
    float gpa;
```

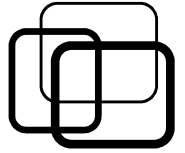
```
    // a, b are UPDATED.
```

```
    input(a, b);
```

```
    // gpa are UPDATED.
```

```
    calcGPA(a, b, gpa);
```

```
}
```



■ Scope:

■ Existing area of variables and functions.

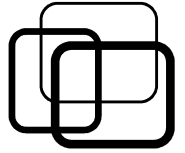
- Global scope: across program.
- Local scope: only in declaration block.

■ Function has global scope.

■ Variable:

- Global variable: declared outside functions (includes `main()`).
 - ➔ Can be used across program.
- Local variable: declared inside a block.
 - ➔ Can be used only in the block.

Function



■ Scope:

```
float S; // Global declarations.
int compute();

int main()
{
    int a = S + compute(); // Local variable in main.
    while (a > 0)
    {
        int b = S + compute(); // Local variable in loop.
    }
}

int compute()
{
    int y = S * 2; // Local variable in function.
}
```

Contents



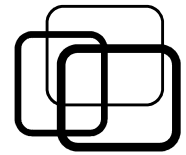
- Function.
- **Multiple-file project.**
- struct.

Multiple-file project



- How do we organize a book?
 - Cannot write in one paper!!
 - ➔ Split into chapters.
 - ➔ Summary at first.
 - ➔ Chapter contents follow.

Multiple-file project



■ Organize C/C++ project:

■ Like a book:

- Chapters ~ source code files.
- Summary ~ main() function.

➔ How to connect multiple source code files?

// File **main.cpp**

```
int main()
{
    input();
    compute1();
    compute2();
    output();
}
```

// File **io.cpp**

```
void input()
{
}

void output()
{
}
```

// File **compute.cpp**

```
int compute1()
{
}

int compute2()
{
}
```

Multiple-file project



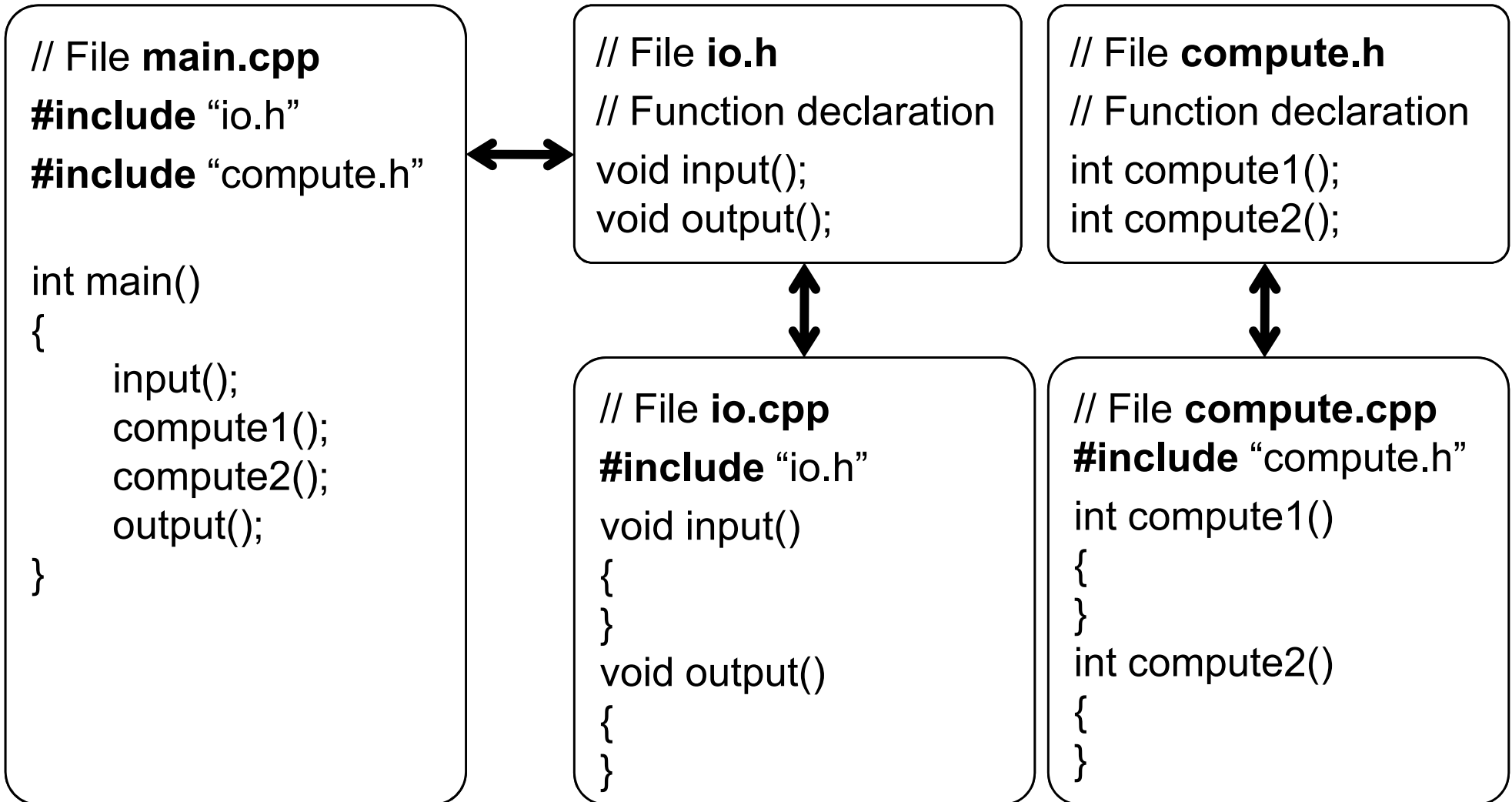
■ Header file:

- Connect source files across project.
- Make code on a file “see” code on another file.
- File extension **.h**.
- Usage:
 - Create header file **.h** for source file **.cpp**.
 - File **.h** contains only **declaration** (global variables/functions).
 - File **.cpp** contains **implementation** of functions.
 - To let A.cpp “see” code in B.cpp
 - ➔ A.cpp **#include** “B.h”

Multiple-file project



■ Header file:

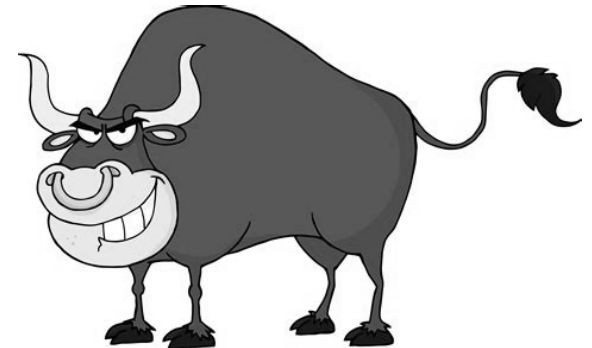


Multiple-file project



■ Divide-conquer a project:

- How to eat a cow?
 - ➔ Split into small parts.
 - ➔ Eat each parts.
- How small is small?
 - ➔ Can be chewed.
- Organize a project:
 - Split into functions and files.
 - Implement each function.
 - Should be < 30 statements.

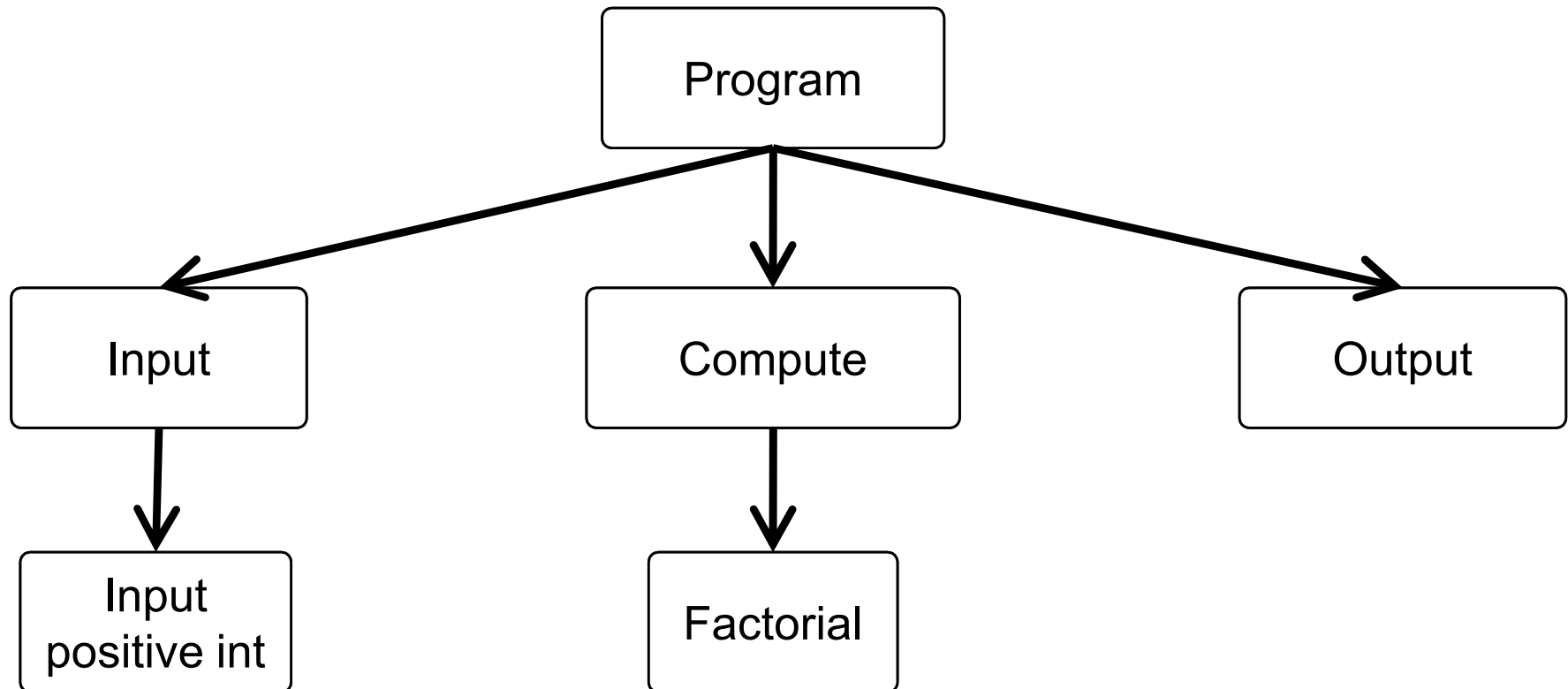


Multiple-file project

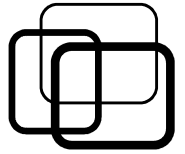


■ Program breakdown tree:

- Enter 3 positive integers $a, b, c \geq 0$.
- Compute and print $S = a! + (b + 1)! + (c + 2)!$.



Multiple-file project



```
// File process1.h
void input(int &a, int &b,int &c);
long compute(int a, int b, int c);
void output(long result);
```

```
// File process2.h
void input_num(int x);
long factorial(int n);
```

```
// File main.cpp
#include "process1.h"

int main()
{
    int    a, b, c;
    long S;

    input(a, b, c);
    S = compute(a, b, c);
    output(S);
}
```

```
// File process1.cpp
#include "process1.h"
#include "process2.h"
void input(int &a, int &b,int &c)
{
    input_num(a);
    input_num(b);
    input_num(c);
}
long compute(int a, int b, int c)
{
    return factorial(a) +
    factorial(b+1) + factorial(c+ 2);
}
void output(long result)
{
    printf("S = %ld", result);
}
```

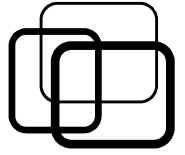
```
// File process2.cpp
#include "process2.h"
#include <stdio.h>
void input_num(int &x)
{
    do {
        printf("Positive integer = ");
        scanf("%d", &x);
    } while (x < 0);
}

long factorial(int n)
{
    long S = 1;
    for ( ; n > 0; n--)
        S = S * n;
    return S;
}
```

Contents



- Function.
- Multiple-file project.
- **struct.**



■ Organize program data:

■ Student information:

- Student id.
- Student name.
- Literature, math.

■ Write program:

- Enter information of a student.
- Print student information with computed gpa.

➔ Identify the inconvenience?

struct



■ C/C++ struct:

- Combine data into one place.
- Compound data-type.
- Declaration:

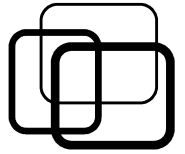
// Syntax 1:

```
struct <struct name>
{
    <struct member 1>;
    <struct member 2>;
    ...
};
```

// Syntax 2:

```
typedef struct
{
    <struct member 1>;
    <struct member 2>;
    ...
} <struct name>;
```

struct



■ C/C++ struct:

■ struct variable:

struct <struct name> <variable>;

// With typedef struct or C++

<struct name> <variable>;

```
struct Student
{
    char id[9];
    char name[50];
    float literature;
    float math;
};

int main()
{
    struct Student s1;
    Student s2; // C++
}
```


struct



■ C/C++ struct:

■ Initialization:

```
struct <struct name> <variable> =  
{ // In declaration order.  
  <member 1 value>,  
  <member 2 value>,  
  ...  
};  
  
struct <struct name> <variable> =  
{ // C99 standard.  
  .<member name> = <value>,  
  ...  
};
```

■ Access struct member:

<variable> . <member name>.

```
int main()  
{  
    struct Student s =  
    {  
        "24127001",  
        "Nguyen Van A",  
        7.5,  
        8.0  
    };  
  
    s.literature = 5.5;  
}
```

struct



■ C/C++ struct:

■ Passing arguments:

- Pass-by-value:
 - ➔ Pass a copy.
 - ➔ Unchanged.
- Pass-by-reference (C++).
 - ➔ Pass the real one.
 - ➔ Can be changed.

```
void add1( struct Student s )
{
    s.literature++;
    s.math++;
}
void add2( Student &s )
{
    s.literature++;
    s.math++;
}
int main()
{
    struct Student s1, s2;
    add1( s1 );
    add2( s2 );
    // s1 unchanged.
    // s2 changed.
}
```

struct



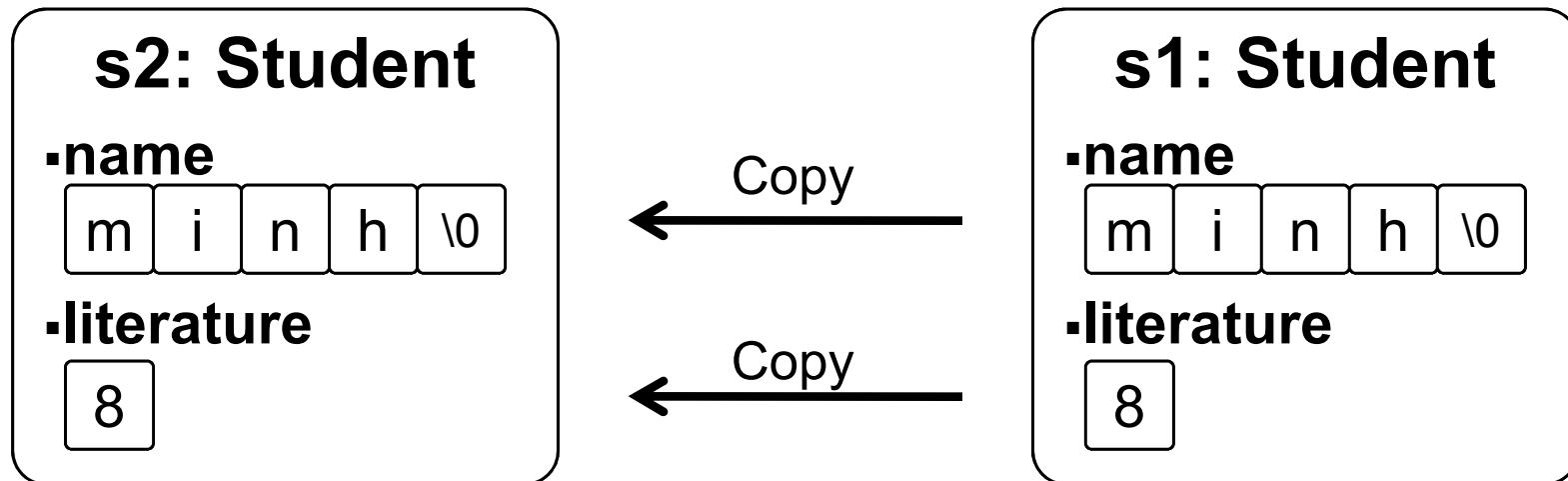
■ C/C++ struct:

■ struct assignment:

- All struct members are copied.
- Array members are copied too!

```
struct Student s1 = { .name = "minh", .literature = 8.0 };
```

```
struct Student s2 = s1;
```



Summary



■ Function:

- A named block of statement can be called anywhere.
- Function structure:
 - Prototype: name, arguments, return type.
 - Body: implementation.
- Passing arguments:
 - pass-by-value.
 - pass-by-reference (C++).

■ Multiple-file project:

- Program ~ book:
 - main() ~ summary.
 - source files ~ chapters.



Summary



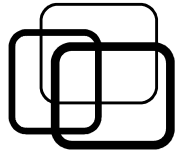
■ Multiple-file project:

- Header file .h: connect code across source files.
- Program breakdown tree:
 - Split program into files and functions.
 - Based on levels of abstraction and reusability.

■ struct:

- Compound data-type.
- Combine data into one place.
- Assignment: all members are copied.





■ Practice 4.1:

Write C/C++ program to find prime numbers:

(organize in functions and multiple-file project)

- Enter a positive integer N (re-enter if invalid).
- Print all prime numbers $\leq N$.

Input format:

Enter a positive integer = 11

Output format:

#1 = 2

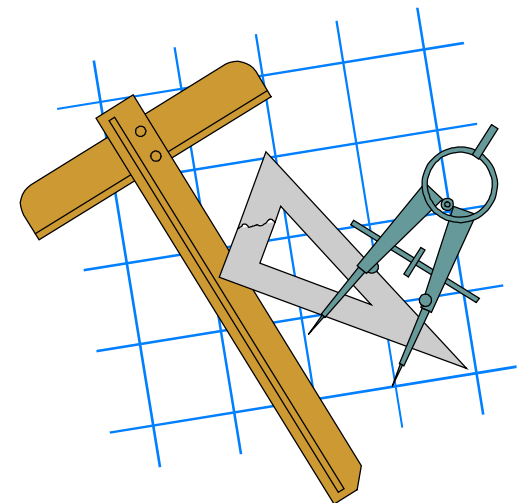
#2 = 3

#3 = 5

#4 = 7

#5 = 11

There are 5 prime numbers.





■ Practice 4.2:

Write C/C++ program to simulate a calculator as follow:

- Enter two integers.
- Enter an operator (+, -, *, /, %).
- Perform the operator on two integers and print result.

Notes: flush the standard input stream after each input.

- C: fgets, or while getchar, C++: cin.getline, or cin.ignore.

Input format:

Enter two integers = 7 5

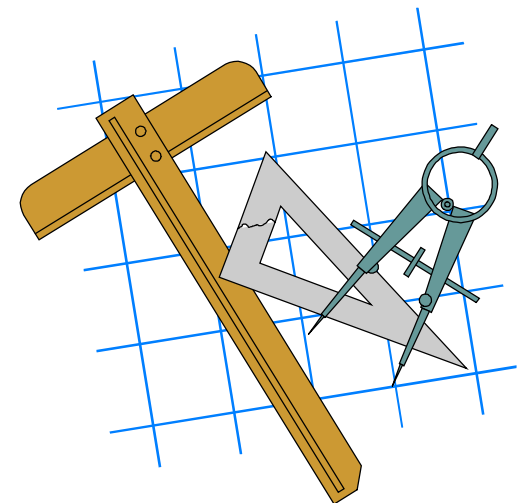
*Enter an opertor (+, -, *, /, %) = +*

Output format (no error):

Result = 12

Output format (divided-by-zero error):

Error: divided by zero.





■ Practice 4.3:

Write C/C++ program to classify a triangle:

(organize in functions and multiple-file project)

- Enter 3 positive real numbers a, b, c (re-enter if invalid).
- Check if a, b, c can form a triangle.
- If yes, print the triangle type.
(normal, right, isosceles, right-isosceles, equilateral).

Input format:

Enter 3 positive real numbers = 3 4 5

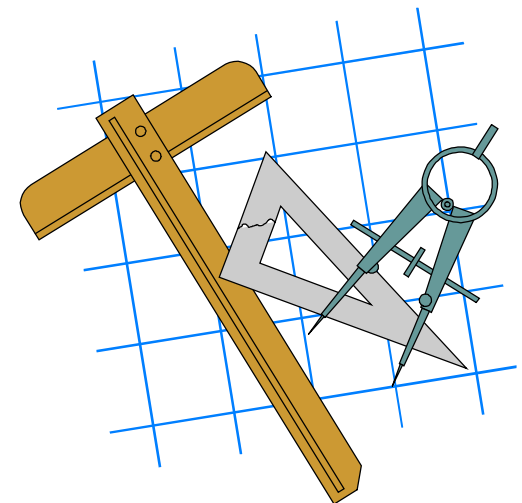
Output format (can form a triangle):

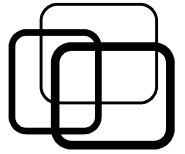
Can form a triangle.

Right triangle.

Output format (cannot form a triangle):

Cannot form a triangle!





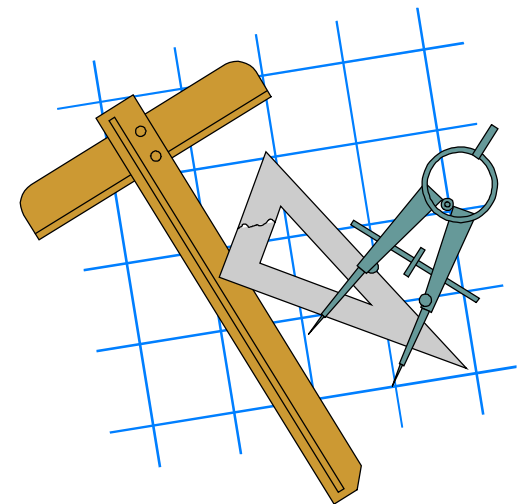
■ Practice 4.4:

Write C/C++ program to simulate a menu as follow:
(organize in functions and multiple-file project):

- Print the menu:

<ul style="list-style-type: none">1. <i>Practice 4.1.</i>2. <i>Practice 4.2.</i>3. <i>Practice 4.3.</i>4. <i>Exit.</i>
<i>Selection (1-4):</i>

- Enter an integer for your selection.
- Selection 1-3:
 - + Execute the selected practice.
 - + Go back to menu.
- Selection 4: exit program.

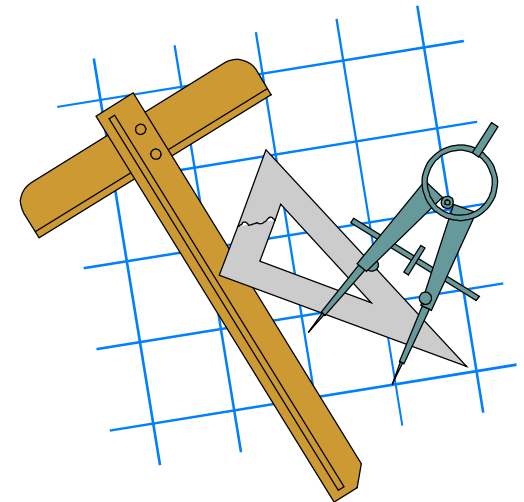




■ Practice 4.5:

Write C/C++ program to operate on fractions:
(organize in functions and multiple-file project):

- Declare struct to represent a fraction.
- Enter 2 fractions.
- Perform the following operations on 2 fractions and print result:
add, multiply, inverse, reduce.

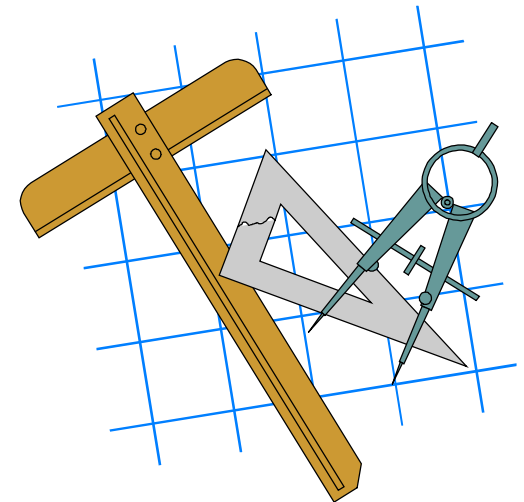


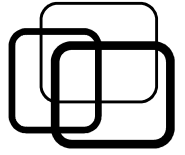


■ Practice 4.6:

Write C/C++ program to operate on students:
(organize in functions and multiple-file project):

- Declare struct to represent a student (stated in the lesson).
- Enter a students.
- Print GPA and their rank:
 - + Excellent: GPA ≥ 8.5 .
 - + Good: GPA ≥ 7.0 .
 - + Fair: GPA ≥ 5.0 .
 - + Failed: GPA < 5.0 .



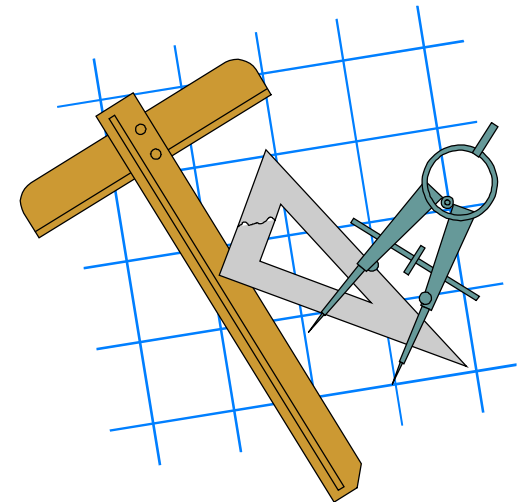


■ Practice 4.7:

Write C/C++ program to operate on triangle:

(organize in functions and multiple-file project):

- Declare structs to represent point (x, y) and triangle (3 points).
- Enter information of a triangle.
- Compute and print triangle perimeter.
- Find and print triangle centroid.



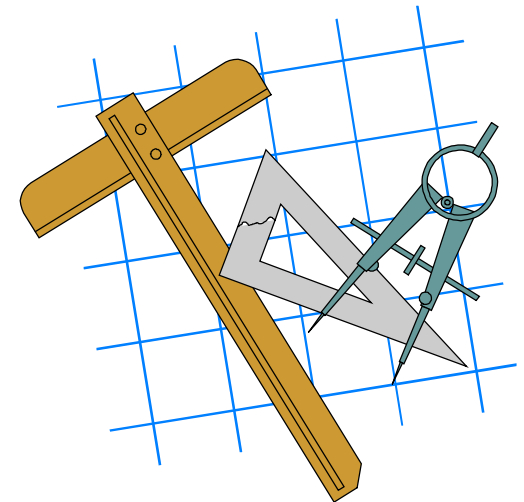


■ Practice 4.8:

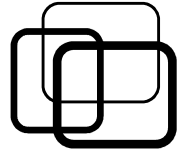
Write C/C++ program to operate on date:

(organize in functions and multiple-file project):

- Declare struct to represent date (day, month, year).
- Enter two date d1 and d2.
- Check if d1 is latest than d2 and print result.
- Print tomorrow date of d1.
- Print yesterday date of d2.



Practice



■ Practice 4.9 (*):

Write compile command for the following projects:

Simple project with multiple folders	Simple project with external libraries	Complex project project 1 uses project 2
project/ bin/ src/ subfolder/	project/ bin/ lib/ libA/ libB/ src/ subfolder/	project/ lib/ libA/ libB/ subproject1/ bin/ src/ subfolder1/ subproject2/ bin/ src/ subfolder2/