โปรแกรมย่อย

(Function, Sub-Program, Program Module)

อ. กฤษณะ สีพนมวัน

ภาควิชาคอมพิวเตอร์

Sep 14, 2009

บทนำ + วิธีคิด

- โปรแกรมคอมพิวเตอร์ส่วนใหญ่ที่ใช้งานจริง มักจะเป็นโปรแกรมขนาดใหญ่ เกินกว่าที่จะเขียนรวมๆ กันในฟังก์ชันหลัก **main()**
- หากแบ่งการทำงานออกเป็นส่วนเล็กๆ หรือ **Module** เมื่อสร้างและทดสอบโปรแกรมย่อยๆ นั้นจนทำงานได้ดี
- แล้วจึงนำโปรแกรมย่อยเหล่านั้นประกอบขึ้นมาเป็นโปรแกรมใหญ่ที่สมบูรณ์ในขั้นตอนสุดท้าย
- จะเป็นการสร้างโปรแกรมที่มีขนาดใหญ่จากโปรแกรมย่อยๆ ซึ่งมีประสิทธิภาพและดำเนินการได้ง่ายกว่าการเขียนโปรแกรมเฉพาะใน ฟังก์ชันหลัก **main()**

คำนิยาม

- ฟังก์ชันหรือโปรแกรมย่อย คือส่วนของโปรแกรมที่สามารถตัดแยกออกเป็นส่วน ๆ เป็นการแบ่งปัญหาใหญ่ที่จะแก้ออกเป็นส่วนเล็ก ๆ ซึ่ง แก้ได้ง่ายกว่า (งงน้อยกว่า)

ฟังก์ชันในภาษาซี

ในภาษาซี โปรแกรมย่อย จะถูกเรียกว่า “ฟังก์ชัน” ซึ่งสามารถแบ่งตามแหล่งที่มาได้ 2 ประเภทคือ

- 1.ฟังก์ชันมาตรฐาน (**Standard Function**)
- 2.ฟังก์ชันที่สร้างขึ้นใหม่โดยโปรแกรมเมอร์ (**Programmer-Defined Function**)

0. ฟังก์ชันแรกในภาษาซี

```
// แบบที่ 1
#include<stdio.h>
void main()
{
   … code ..
}
```

```
// แบบที่ 2
#include<stdio.h>
int main(void)
{
   … code ..
   return 0;
}
```

ในภาษาซีเริ่มต้นทำงานในฟังก์ชัน **main()** และในภาษาซี มีฟังก์ชัน(ชื่อ) **main()** เพียงฟังก์ชันเดียว

1. ฟังก์ชันมาตรฐาน

- ฟังก์ชันมาตรฐานในภาษาซีจะอยู่ในไลบรารีมาตรฐาน (Standard Library) ซึ่งประกอบด้วยฟังก์ชันต่างๆ มากมาย ไม่ว่าจะใช้สำหรับการคำนวณทางคณิตศาสตร์ การจัดการกับข้อความ การจัดการกับ input/output และอื่นๆ ซึ่งจะทำให้งานของโปรแกรมเมอร์ง่ายขึ้น
- โดยการใช้งานฟังก์ชันประเภทนี้จะต้องรวม (#include) ไลบรารีที่ต้องการใช้งาน

เช่น ฟังก์ชัน printf(), scanf(), clrscr(), getch()

โดยหากต้องการใช้ getch() เราต้อง include conio.h

2. ฟังก์ชันที่สร้างขึ้นใหม่โดยโปรแกรมเมอร์

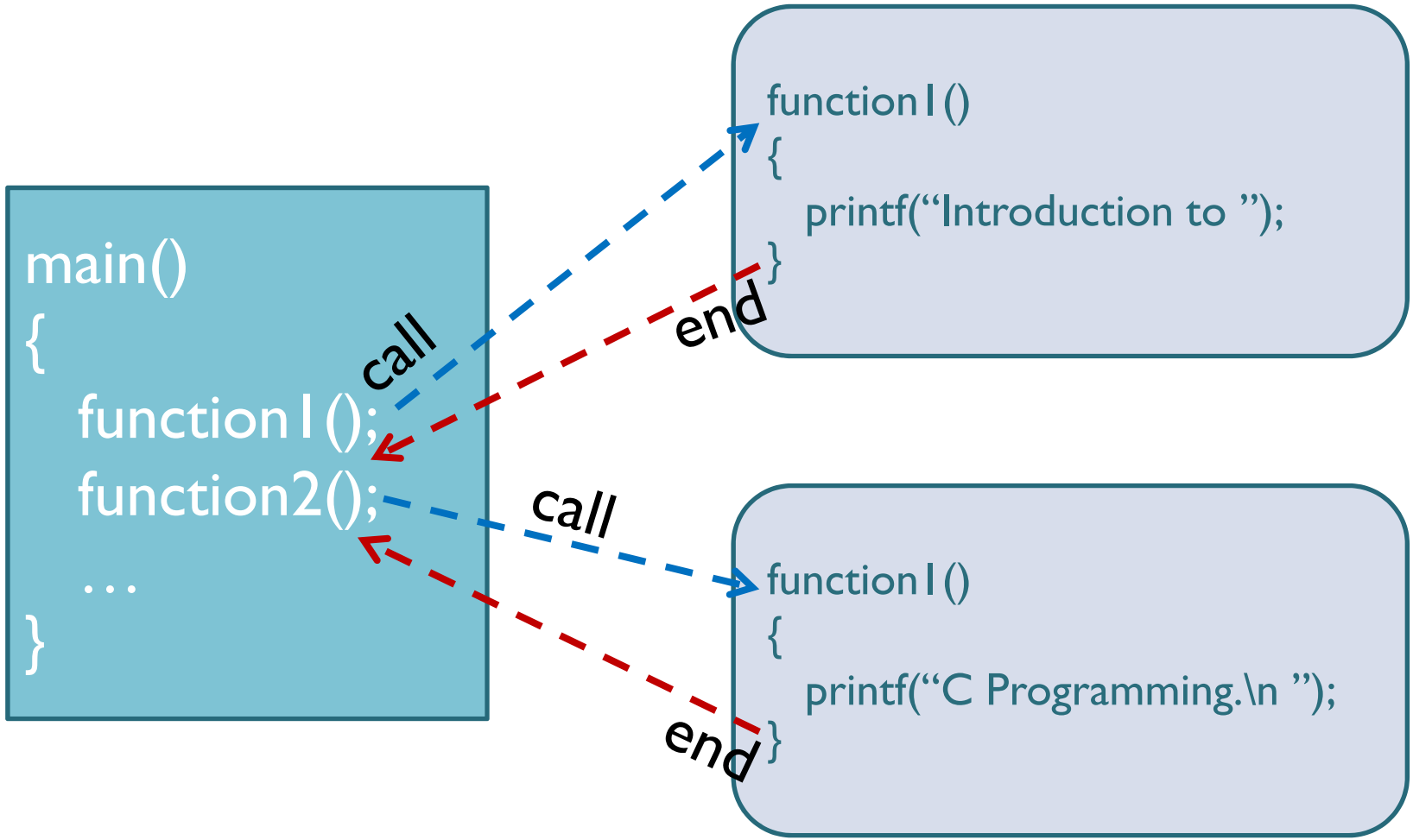
- โปรแกรมเมอร์สามารถเขียนฟังก์ชันเพื่อนิยามการทำงานที่จะเรียกใช้ในส่วนต่างๆของโปรแกรม โดยฟังก์ชันการทำงานดังกล่าวจะถูกเขียนไว้ในฟังก์ชันเพียงครั้งเดียวเท่านั้น
- แต่สามารถเรียกใช้งานได้หลายครั้ง
- ตัวแปรที่ประกาศหรือคำสั่งที่เรียกในฟังก์ชันใดๆมีขอบเขตการใช้งานอยู่ในฟังก์ชันนั้น ๆ เท่านั้น
- นั่นหมายความว่าฟังก์ชันอื่น ๆ รวมทั้งฟังก์ชัน **main()** จะไม่ทราบการทำงานภายในหรือคำสั่งต่าง ๆ ในฟังก์ชันนั้น
- หรือพูดอีกอย่างได้ว่าฟังก์ชันแต่ละฟังก์ชันจะไม่ทราบการทำงานภายในของฟังก์ชันอื่น
- เพียงแต่จะสามารถเรียกใช้งานฟังก์ชันนั้นได้เท่านั้น

การไม่เขียนเป็นฟังก์ชัน

```
#include<stdio.h>
main()
{
   printf(“Introduction to ”);
   printf(“C Programming.\n”);
   …
}

// ผลลัพธ์   Introduction to C Programming.
```

เขียนฟังก์ชันและใช้งานในโปรแกรม



// ผลลัพธ์ Introduction to C Programming.

ข้อควรจำเมื่อจะใช้งานฟังก์ชัน

- สร้างแล้วควรนำไปใช้งานด้วย
- เมื่อทำเป็นฟังก์ชันแล้วจะช่วยทำให้แก้ปัญหาได้ง่ายขึ้น ไม่ใช่ทำให้แก้ปัญหาได้ยากขึ้น
- ขอบเขตของตัวแปร : ตัวแปรที่กำหนดในฟังก์ชัน มีข้อมูลและสามารถใช้งานได้เฉพาะภายในฟังก์ชันเท่านั้น
- ฟังก์ชันหนึ่ง ๆ ควรทำงานเพียงงานเดียว
- ฟังก์ชันจบเมื่อสิ้นสุดคำสั่งใน **block** หรือพบคำสั่ง **return**

การนิยามฟังก์ชัน

Syntax:

```
return_type  function_name( parameters )
{
   // code ของฟังก์ชัน
}
```

ตัวอย่าง

```
void  function1()
{
   …
}
```

```
int  function2(int p)
{
   …
   return ?;
}
```

```
int  function3(int p1, int p2)
{
   …
   return ?;
}
```

การสร้างฟังก์ชัน

หัวข้อที่กล่าวถึง :

- Function prototype
- ฟังก์ชันอย่างง่าย
- ฟังก์ชันที่รับพารามิเตอร์
- ฟังก์ชันที่รับพารามิเตอร์และส่งค่ากลับ

Function Prototype

- คือ การระบุรูปแบบของฟังก์ชันตาม **syntax** เพื่อบอกให้ผู้ใช้หรือตัวโปรแกรมเมอร์เองทราบถึงวิธีการใช้งานฟังก์ชันที่สร้าง
- การนิยาม **prototype** ไม่ต้องเขียนส่วนของการทำงาน
- โดย
  - **จำเป็น** หากสร้างหรือกำหนดฟังก์ชันนั้น ๆ ไว้หลังฟังก์ชัน **main**
  - **ไม่จำเป็น** หากสร้างหรือกำหนดฟังก์ชันนั้น ๆ ไว้ก่อนฟังก์ชัน **main**
- **Syntax :**
  - **void function1();** // ประกาศเพียงเท่านี้ อย่าลืม **;**

Function Prototype (ต่อ)

จำเป็น !!

```
#include <stdio.h>

void function1();

int main(void)
{

   function1();

}

void function1()
{
   …
}
```

prototype

ไม่จำเป็น !!

```
#include <stdio.h>

void function1()
{
   …
}

int main(void)
{

   function1();

}
```

ฟังก์ชันอย่างง่าย

- ฟังก์ชันแบบนี้ ใช้ในกรณีที่เราสามารถแบ่งแยกส่วนของโปรแกรมออกมาจากโปรแกรมหลักได้อย่างอิสระ
- การทำงานในฟังก์ชันไม่มีการรับ – ส่ง ค่าของตัวแปร หรือการคำนวณใด ๆ ที่เกี่ยวข้องกับโปรแกรมหลัก
- **ตัวอย่าง**

```
void function1(void)
{
   printf("This print of function1");
}
```

```
void function2(void)
{
   int i, x = 10;
   for(i=0; i<x; i++)
      printf("%d", x);
}
```

ฟังก์ชันที่รับพารามิเตอร์

- พารามิเตอร์จะถูกส่งผ่านไปยังฟังก์ชันโดยผ่านทางตัวแปรที่กำหนดภายในเครื่องหมายวงเล็บ () หลังชื่อฟังก์ชัน
- พารามิเตอร์ที่รับเข้ามาเป็นค่าของข้อมูลที่ส่งมาจากโปรแกรมหลัก และต้องเป็น **data_type** ชนิดเดียวกัน
- **ตัวอย่าง**

รับ parameter แบบ int เท่านั้น

```
void print_x(int x)
{
    int i;
    for(i=0; i< x; i++)
      printf(" X ");
}
```

```
int main(void)
{
    int y = 10,  z = 15;
    print_x( y );
    print_x( z );
    print_x( 20.5 );
    return 0;
}
```

ส่ง parameter ชื่อ y ชนิด int

ส่ง parameter ชื่อ z ชนิด int

ส่ง parameter เป็นค่าคงที่ 20.5 ไม่ใช่ int

ฟังก์ชันที่รับพารามิเตอร์ (ต่อ)

- เราสามารถส่งพารามิเตอร์เข้าไปยังฟังก์ชันได้หลาย ๆ ค่า
- โดยกำหนดจำนวนพารามิเตอร์ภายในเครื่องหมายวงเล็บ () แบ่งด้วยเครื่องหมาย **comma**
- และในการส่งค่าต้องส่งให้ถูกต้องตามลำดับ
- **ตัวอย่าง**

รับ parameter 3 ตัว

```
void print_char(int x, float y, char z)
{
    int i;
    for(i=0; i< x; i++)
      printf("%c  %.2f", z , y);
}
```

```
int main(void)
{
    int y = 10,  z = 15;
    char a = 'A';
    float b = 2.5f;
    print_char( y, b, a);
    print_char(2, 20.5, 'X');
    return 0;
}
```

ส่ง parameter ด้วยตัวแปร

ส่ง parameter ด้วยค่าคงที่

ฟังก์ชันที่รับพารามิเตอร์และส่งค่ากลับ

- หมายถึงหลังจากทำงานภายในฟังก์ชัน ซึ่งอาจจะเป็นการคำนวณค่าจากพารามิเตอร์ที่รับเข้ามา เสร็จเรียบร้อยแล้ว จะส่งค่าที่คำนวณได้กลับไปยังโปรแกรมหลัก
- ส่งค่ากลับได้เพียง 1 ค่าเท่านั้น
- **ตัวอย่าง**
  - โจทย์ : จงเขียนฟังก์ชันหาผลบวกของตัวเลข 2 ตัว
  - วิธีทำ :
    - ตั้งชื่อฟังก์ชันให้สื่อถึงการทำงานภายใน
    - ต้องใช้พารามิเตอร์กี่ตัว ชนิดใด **int float double char ?**
    - ส่งค่ากลับเป็นชนิดใด
    - คำนวณอย่างไร
    - ใช้งานในโปรแกรมหลักอย่างไร

ตย. ฟังก์ชันหาผลบวกของเลข 2 ตัว

```
int add(int a, int b)
{
   return a+b;
}
```

```
float sum(float a, float b)
{
   int c;
   c = a + b;
   return c;
}
```

```
#include<stdio.h>
int main(void)
{

   int x = 2, y = 3;
   printf(" %d ",  x+y);
   printf(" %d ",  add(x, y));
   printf(" %f  ",  sum(12.5, 13.5));

   printf(" %d ", add(5,  add(20, 30)));

   return 0;
}
```

Test 1: simple functions

1. ฟังก์ชันแปลงค่าเงินจาก **baht** เป็น **dollar**
2. ฟังก์ชันคิดค่าสินค้ารวม **vat**
3. ฟังก์ชันเลขยกกำลัง
4. ฟังก์ชันคำนวณจำนวนวินาที

การใช้งานในโปรแกรมหลัก

```
1. printf("500 bahts = %f  dollars", exchange( 500,  0.02896 ));
2. printf("you must pay  %f  Bahts", 100 + vat( 100,  7));
3. printf(" 2^5 = %d ", power( 2, 5 ));
4. printf("2 hours 2 minutes 59 seconds =  %d ticks",  ticks( 2, 2, 59 ));
```

เฉลย

```
double  exchange(int a, double b)
{
      return a * b;
}

double  vat(int a, int b)
{
      return a + ( a*b / 100);
}

int  power(int a, int b)
{
   int i, x;
   x = 1;
   for(i=0; i<b; i++)
      x = x * a;
   return x;
}

int ticks(int h, int m, int s)
{
    return (h * 3600) + (m * 60) + s;
}
```