2110104: Computer Programming

Function (advance)

Jessada Thutkawkorapin

DEPT. OF COMPUTER ENGINEERING
CHULALONGKORN UNIVERSITY

ทบทวนเนื้อหา

กำหนดชนิดตัวแปร
หากต้องมีการคืนค่า
void ในที่นี้คือไม่คืนค่า
return type

ชื่อฟังก์ชัน
function name

ส่วนที่ใช้กำหนดว่าฟังก์ชันจะรับค่าอะไรบ้าง
หากไม่มีสามารถกำหนดเป็น ( ) ได้
parameters

```
void func_name (int var_name) {
    // function body (code)
}
```

ส่วนเนื้อหาของฟังก์ชันจะอยู่ภายใน { }
statements

Function parameters

```
# include <iostream>
using namespace std;
void swap_int(int x1, int x2) {
    int tmp = x1;
    cout << "In func (1): " << x1 << " " << x2 << endl;
    x1 = x2;
    x2 = tmp;
    cout << "In func (2): " << x1 << " " << x2 << endl;
}
int main() {
    int a=3, b=4;
    cout << "At (1): " << a << " " << b << endl;
    swap_int(a, b);
    cout << "At (2): " << a << " " << b << endl;
}
```

จะแสดงผล

```
At (1): 3 4
In func (1): 3 4
In func (2): ? ?
At (2): ? ?
```

Function parameters (cont)

```
# include <iostream>
using namespace std;
void swap_int(int x1, int x2) {
    int tmp = x1;
    cout << "In func (1): " << x1 << " " << x2 << endl;
    x1 = x2;
    x2 = tmp;
    cout << "In func (2): " << x1 << " " << x2 << endl;
}
int main() {
    int a=3, b=4;
    cout << "At (1): " << a << " " << b << endl;
    swap_int(a, b);
    cout << "At (2): " << a << " " << b << endl;
}
```

จะแสดงผล

```
At (1): 3 4
In func (1): 3 4
In func (2): 4 3
At (2): 3 4
```

คำถาม

```
# include <iostream>
using namespace std;
void swap_int(int &x1, int &x2) {
    int tmp = x1;
    cout << "In func (1): " << x1 << " " << x2 << endl;
    x1 = x2;
    x2 = tmp;
    cout << "In func (2): " << x1 << " " << x2 << endl;
}
int main() {
    int a=3, b=4;
    cout << "At (1): " << a << " " << b << endl;
    swap_int(a, b);
    cout << "At (2): " << a << " " << b << endl;
}
```

จะแสดงผล

```
At (1): 3 4
In func (1): 3 4
In func (2): ? ?
At (2): ? ?
```

เฉลย

```
# include <iostream>
using namespace std;
void swap_int(int &x1, int &x2) {
    int tmp = x1;
    cout << "In func (1): " << x1 << " " << x2 << endl;
    x1 = x2;
    x2 = tmp;
    cout << "In func (2): " << x1 << " " << x2 << endl;
}
int main() {
    int a=3, b=4;
    cout << "At (1): " << a << " " << b << endl;
    swap_int(a, b);
    cout << "At (2): " << a << " " << b << endl;
}
```

(int x1) pass by value

(int &x1) pass by reference

จะแสดงผล

```
At (1): 3 4
In func (1): 3 4
In func (2): 4 3
At (2): 4 3
```

Pass by value

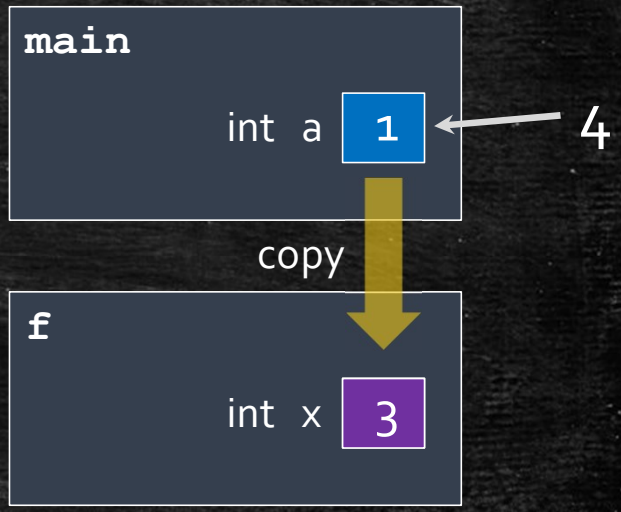
```
# include <iostream>
using namespace std;
void f(int x) {
    x = 3;
}
int main() {
    int a = 1;
    cout << "At (1): " << a << endl;
    f(a);
    cout << "At (2): " << a << endl;
}
```

3 → `x = 3;`

1 → `int a = 1;`

2 → `f(a);`

4 → `cout << "At (2): " << a << endl;`



Pass by reference

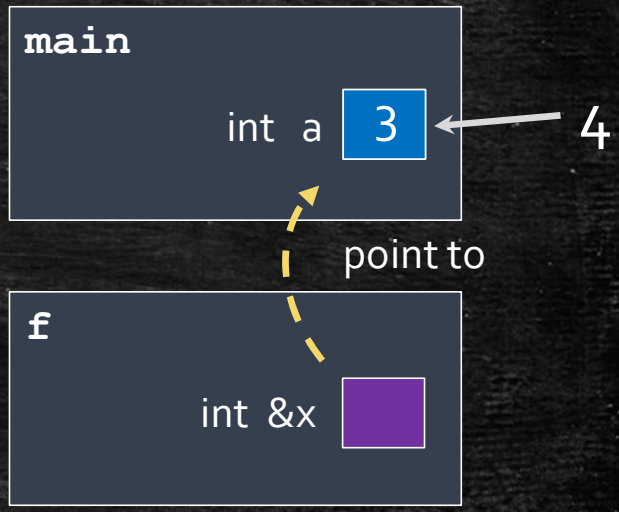
```
# include <iostream>
using namespace std;
void f(int &x) {
    x = 3;
}
int main() {
    int a = 1;
    cout << "At (1): " << a << endl;
    f(a);
    cout << "At (2): " << a << endl;
}
```

3 → `x = 3;`

1 → `int a = 1;`

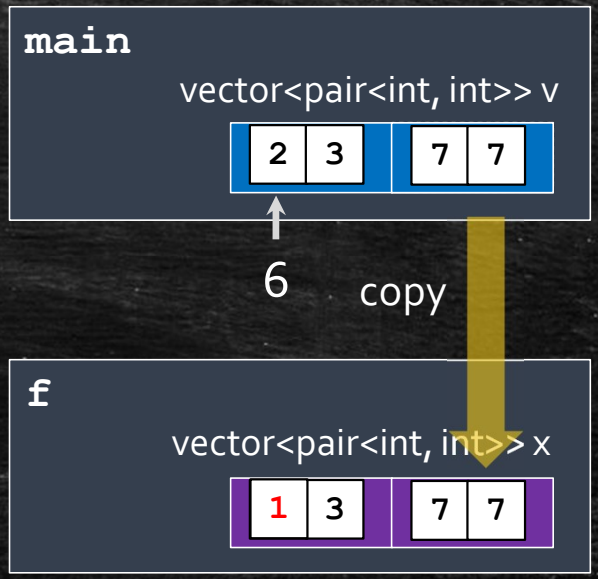
2 → `f(a);`

4 → `cout << "At (2): " << a << endl;`



Pass by value (vector)

```
# include <iostream>
# include <vector>
using namespace std;
void f(vector<pair<int, int>> x) {
5 ->    x[0].first = 1;
}
int main() {
1 ->    vector<pair<int, int>> v;
2 ->    v.push_back(make_pair(2, 3));
3 ->    v.push_back(make_pair(7, 7));
        cout << v[0].first << endl;
4 ->    f(v);
6 ->    cout << v[0].first << endl;
}
```



สรุป pass by value vs reference

- pass by value
  - หน่วยความจำแยกกันระหว่างผู้ส่งกับผู้รับค่า
    - การแก้ไขค่าในฟังก์ชันของผู้รับค่าจะไม่กระทบกับผู้ส่งค่า
    - ต้องมีการก๊อปข้อมูลทั้งก้อนจากผู้ส่งมายังผู้รับ (มีผลมากโดยเฉพาะกับข้อมูลขนาดใหญ่)
- pass by reference
  - ผู้ส่งกับผู้รับแชร์ที่เก็บข้อมูลที่เดียวกัน
    - การแก้ไขค่าในฟังก์ชันของผู้รับค่าจะทำให้ค่าที่เก็บที่ผู้ส่งค่าเปลี่ยนไปด้วย
    - ไม่มีขั้นตอนของการก๊อปข้อมูล

Recursive function

ฟังก์ชันเรียกซ้ำ

Recurrence relation (ความสัมพันธ์เกิดซ้ำ)

- A recurrence relation is an equation that defines a sequence based on a rule that gives the next term as a function of the previous term(s)
- Ex. Fibonacci sequence

$$F(n) = F(n-1) + F(n-2)$$

F0	F1	F2	F3	F4	F5	F6	F7	F8	F9
0	1	1	2	3	5	8	13	21	34

Recursive function template

```
# include <iostream>
using namespace std;
int f(int n) {
    if ( end_condition ) {
        return something;
    }
    // process some other things
    //

    // then call itself with new value
    return n + f(n-1);
}
int main() {
    cout << f(5) << endl;
}
```

Ex1: factorial

$F(n) = n \times F(n-1)$
$F(1) = 1$

```
# include <iostream>
using namespace std;
int fac(int n) {
    if (n == 1) return 1;
    return n * fac(n-1);
}
int main() {
    cout << fac(5) << endl;
}
```

จะแสดงผล

```
120
```

Ex2: fibonacci

F(n) = F(n-1) + F(n-2)
F(1) = 1
F(0) = 0

F0	F1	F2	F3	F4	F5	F6	F7	F8	F9
0	1	1	2	3	5	8	13	21	34

```
# include <iostream>
using namespace std;
int fibo(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibo(n-1) + fibo(n-2);
}
int main() {
    cout << fibo(9) << endl;
}
```

จะแสดงผล

```
34
```

สรุป recursive function

- ส่วนประกอบสำคัญ
  - เงื่อนไขการหยุด
  - เรียกตัวมันเองด้วยค่าที่ต่างออกไป
- ข้อดี
  - เขียนได้ง่าย ตรงไปตรงมา
- ข้อเสีย
  - หาบั๊กยาก
  - ช้าในแง่ algorithm (เช่น Fibonacci)
  - ช้าในแง่หาก pass by value ของข้อมูล parameters ที่ค่อนข้างใหญ่ ก็จะมีจำนวนครั้งที่ต้องก๊อปเท่ากับจำนวนการเรียกตัวมันเอง (หมดเวลาไปกับการก๊อปข้อมูล)
  - มักจะมีทางเลือกที่ดีกว่าที่เขียนได้ด้วยการวนลูปธรรมดา