Array

A list of same type variable

Nattee Niparnan

Department of Computer Engineering
Chulalongkorn University

ทำงานแบบเดิมซ้ำ ๆ

- อาเรย์คือตัวแปรประเภทเดียวกันหลาย ๆ ตัวที่มาอยู่รวมกันเป็นรายการ
  - ในรายการมีช่องหลาย ๆ ช่อง แต่ละช่องคือตัวแปรหนึ่งตัวประเภทเดียวกัน
- ประกาศครั้งเดียวได้ตัวแปรมาหลายตัวพร้อมกัน
- ใช้งานแต่ละตัวได้สะดวก

ตัวอย่าง

- `int a[10];`
- ได้ตัวแปรมา 11 ตัว
  - `a` เป็น อาเรย์
  - `a[0]`, `a[1]`, ..., `a[9]` เป็น `int`
- เรียกใช้แต่ละตัวด้วย `a[x]` ได้ โดยที่ `x` เป็น expression แบบจำนวนเต็ม
  - ทำให้เขียนโปรแกรมใช้งานแต่ละตัวได้สะดวก
- ลองเทียบกับ `int b1,b2,b3,b4,b5,b6;`
  - ประกาศยาก
  - เรียกใช้ก็ยาก

คุ้น ๆ ว่าเหมือนกับ string มั้ย?

จริง ๆ แล้ว เรามองว่า string คือ อาเรย์ของ char ก็พอได้

```cpp
#include <iostream>
using namespace std;

int main() {
  int a[6];
  int b1,b2,b3,b4,b5,b6;

  for (int i = 0;i < 6;i++) {
    a[i] = i*10;
    //bi = i*10 //  ทำไม่ได้
  }
  a[1] = a[5] = -1;
  cout << a[0] + a[4] << endl;
}
```

[ ] มีชื่อเรียกเพราะ ๆ ว่า subscription operator และถูกมองเป็น operator แบบหนึ่งเหมือนกัน

ของข้างใน [ ] จะต้องเป็นจำนวนเต็ม

การประกาศอาเรย์

- ใช้รูปแบบ

```
ประเภทตัวแปรแต่ละช่อง ชื่อ[จำนวนช่อง];
```

- เช่น
  - `int a[6];`
  - `double d[10000];`
  - `bool b[10000];`
- ตัวแปรที่ได้คือ `ชื่อ[0]` ถึง `ชื่อ[จำนวนช่อง-1]`
  - `a[0]` ถึง `a[5]`

ข้อควรระวังการใช้ [ ]

- เหมือนกับ string คือการเรียกใช้ a[x] นั้น x ต้องเป็นหมายเลขช่องที่มีอยู่จริง ๆ
- เช่น ประกาศไว้ 6 ช่อง ก็จะเรียกใช้ได้เฉพาะช่อง [0] ถึง [5]
  - เรียก [-2] หรือ [10000] ไม่ได้
  - C++ อาจจะไม่เตือนตอน compile แต่พอเอาไปใช้งานจริง มีโอกาสที่โปรแกรมจะทำงานผิดพลาดได้
  - ผล X ใน grader นั้น หลาย ๆ กรณีเกิดจากการพยายามอ่านหรือเขียนช่องที่ไม่มีในอาเรย์

```
int a[6];
int b[10];

a[-2] = 20;
b[10000];

cout << b[10] << " " << a[100] << endl;
```

ผิด!!!

ใช้ [ ] นอกเหนือจากช่องที่มี

วิธีการหาจำนวนช่องของอาเรย์

1. จำเอา (ดู a)
   - สร้างค่าคงที่มาเก็บค่าความยาว

2. คำนวณเอา (ดู b)

   คำนวณจำนวนช่องจากจำนวนพื้นที่ทั้งหมดที่ใช้หารด้วยขนาดของแต่ละช่อง

```cpp
#include <iostream>
using namespace std;

const int n = 10;

int main() {
  int a[n];
  for (int i = 0;i < n;i++)
    a[i] = 1;

  int b[10];
  for (int i = 0;i < sizeof(b) / sizeof(int);i++)
    b[i] = 2;

  cout << "Size of a " << sizeof(a) << endl;
  cout << "Size of b " << sizeof(b) << endl;
  cout << "Size of a[i] " << sizeof(a[1]) << endl;
}
```

const เป็นการระบุว่าตัวแปรนี้จะไม่มีการเปลี่ยนแปลงค่า (เป็นค่าคงที่ หรือ constant)

sizeof จะคำนวณพื้นที่หน่วยความจำที่ตัวแปรใช้ (หน่วยเป็น byte)

output

```
Size of a 40
Size of b 40
Size of a[i] 4
```

ลองดูขนาดอื่นบ้าง

```
bool c[5];
cout << "Size of c " << sizeof(c)
  << " each slot is " << sizeof(c[0]) << endl;

double d[100];
cout << "Size of c " << sizeof(d)
  << " each slot is " << sizeof(d[0]) << endl;
```

output

```
Size of c 5 each slot is 1
Size of c 800 each slot is 8
```

- double 8 bytes
- bool 1 byte

การส่งอาเรย์ไปยังฟังก์ชัน

- พารามิเตอร์ที่เป็นอาเรย์ ให้ใส่ [ ] ต่อท้าย โดยไม่จำเป็นต้องระบุขนาด
  - ระบุก็ได้ แต่ไม่มีผลแตกต่าง
  - และการเรียกใช้ sizeof กับอาเรย์ที่รับมาผ่านฟังก์ชันจะใช้ไม่ได้
- ถ้าอยากทราบขนาดอาเรย์ ต้องระบุพารามิเตอร์เพิ่มเติม

```
int get_max(int a[], int n) {
  int max_pos = 0;
  for (int i = 0;i < n;i++) {
    if (a[i] > a[max_pos])
      max_pos = i;
  }
  return a[max_pos];
}
```

```
void test1(int a[3]) {
  cout << "test 1 size = " << sizeof(a) << endl;
  for (int i = 0;i < 5;i++) cout << a[i] << " ";
  cout << endl;
}

void test2(int a[100]) {
  cout << "test 2 size = " << sizeof(a) << endl;
  for (int i = 0;i < 5;i++) cout << a[i] << " ";
  cout << endl;
}

int main() {
  int a[3];
  int b[7];
  for (int i = 0;i < 3;i++) a[i] = i;
  for (int i = 0;i < 7;i++) b[i] = i*10;
  test1(a); test2(a);
  test1(b); test2(b);
}
```

sizeof ของการเรียกทั้ง 4 ครั้งได้ค่าเดียวกันหมด

output

```
test 1 size = 8
0 1 2 7 3
test 2 size = 8
0 1 2 7 3
test 1 size = 8
0 10 20 30 40
test 2 size = 8
0 10 20 30 40
```

การแก้ไขอาเรย์ในฟังก์ชัน

```
void print(char c[],int n) {
  for (int i = 0;i < n;i++)
    cout << c[i];
  cout << endl;
}

void fill(char a[],char value, int begin, int end) {
  while (begin < end)
    a[begin++] = value;
}

void test(int a) {
  a = 10;
}

int main() {
  int a = 5;
  test(a);
  cout << "a is " << a << endl;

  char c[10];
  for (int i = 0;i < 10;i++) c[i] = 'a';
  print(c,10);
  fill(c,'x',3,6);
  print(c,10);
}
```

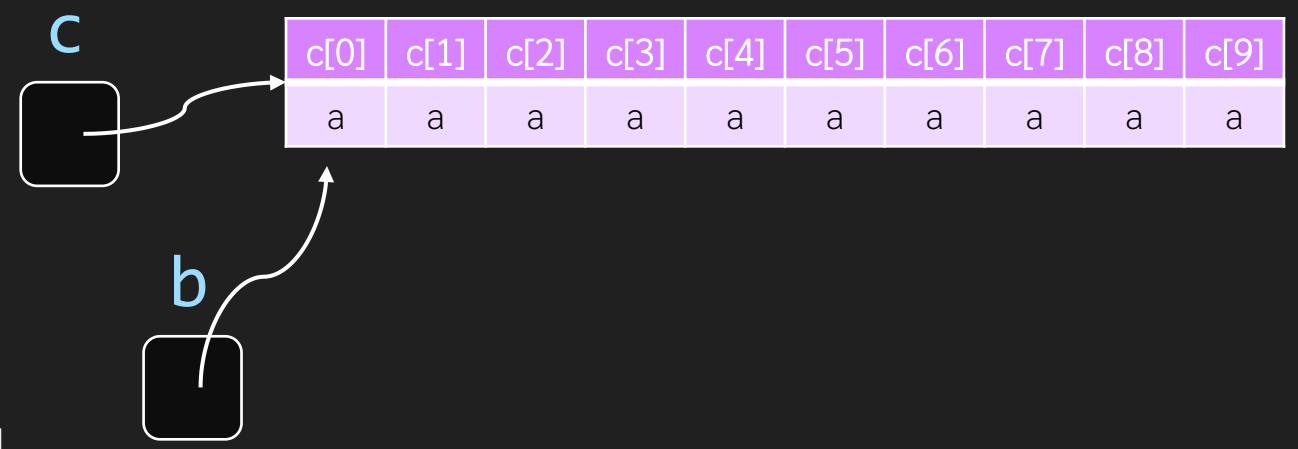
- ระวัง! การส่งอาเรย์เป็นพารามิเตอร์นั้น การแก้ไขอาเรย์ในฟังก์ชันจะทำให้อาเรย์ที่ส่งเข้าไปเปลี่ยนแปลงด้วย
  - สังเกตเปรียบเทียบระหว่าง `fill` กับ `test`

output

```
a is 5
aaaaaaaaaa
aaaxxxaaaa
```

เหตุผลที่อาเรย์เปลี่ยนค่าได้ในฟังก์ชัน

- การส่งพารามิเตอร์ในฟังก์ชันนั้น โดยปรกติจะเป็นการ “ทำสำเนา” ของค่าที่ส่งเข้าไป
- แต่ตัวแปรอาเรย์นั้นจริง ๆ แล้วเก็บ “ที่อยู่” ของช่องข้อมูลหลาย ๆ ช่อง
- การเรียกฟังก์ชันที่มีพารามิเตอร์เป็นอาเรย์ ตัวแปรในฟังก์ชัน กับ นอกฟังก์ชันเป็นคนละตัวกันก็จริง แต่เก็บค่าเดียวกันคือ “ที่อยู่” ของก้อนอาเรย์เดียวกัน




```
void fill(char b[],char value, int begin, int end) {
  while (begin < end)
    b[begin++] = value;
}

int main() {
  char c[10];
  for (int i = 0;i < 10;i++) c[i] = 'a';
  print(c,10);
  fill(c,'x',3,6);
  print(c,10);
}
```

ประกาศอาเรย์พร้อมกำหนดค่าเริ่มต้น

```
void print(int c[],int n) {
  for (int i = 0;i < n;i++)
    cout << c[i] << " ";
  cout << endl;
}

int main() {
  int z[6];
  int a[3] = {1,2,3};
  int b[5] = {1,2};
  int c[] = {11,22,33,44};
  //int c[3] = {10,20,30,40};
  print(z,6);
  print(a,3);
  print(b,5);
  print(c,4);
}
```

z ไม่ถูกกำหนดค่าเริ่มต้น

b ทั้ง 5 ช่องถูกกำหนดค่าเริ่มต้น โดยสามช่องหลังเป็น 0

c มี 4 ช่อง

ทำไม่ได้ เพราะจะกำหนดค่าเริ่มต้นมากเกินช่องที่มี

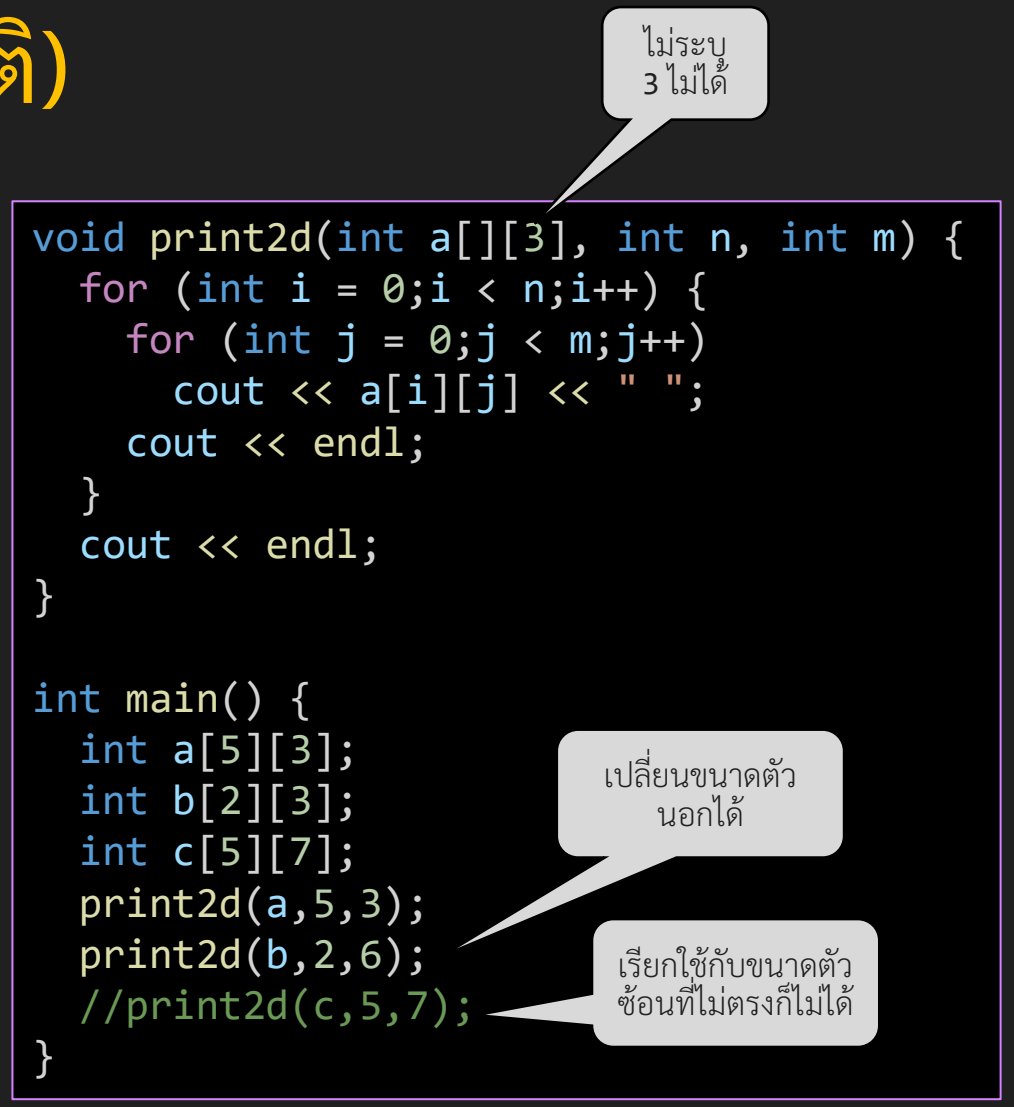
- กำหนดค่าเริ่มต้นให้กับอาเรย์ตอนประกาศได้โดยใช้รูปแบบ
- `{ค่าช่อง 0, ค่าช่อง 1 , ค่าช่อง 2,...}`
- หากอาเรย์มีช่องมากกว่าที่กำหนดค่าเริ่มต้น ช่องที่เหลือเหล่านั้นจะเป็นค่าเริ่มต้นปรกติของตัวแปรนั้น (มักจะเป็น 0)
- สามารถไม่ระบุขนาดได้ โดยขนาดจะเท่ากับรายการที่กำหนดค่าเริ่มต้นพอดี

output

```
8 0 50 0 16849040 0
1 2 3
1 2 0 0 0
11 22 33 44
```

อาเรย์ซ้อนอาเรย์ (อาเรย์หลายมิติ)

- ใช้วิธีกำหนดขนาดต่อท้ายไปเรื่อย ๆ
- การส่งเป็นพารามิเตอร์ของฟังก์ชัน ต้อง ระบุขนาดของอาเรย์ตัวที่ซ้อนอยู่ภายในด้วย
  - ตัวนอกสุดไม่ต้องระบุก็ได้ (และอาเรย์ที่ส่งมามีขนาดต่างกันได้)



ลอง 3 มิติ

```
int test3d(int a[][5][10],int n1, int n2, int n3) {
  int sum = 0;
  for (int i = 0;i < n1;i++)
    for (int j = 0;j < n2;j++)
      for (int k = 0;k < n3;k++)
        sum+=a[i][j][k];
  return sum;
}

int main() {
  int p[4][5][10];
  int q[100][5][10];
  int r[4][99][10];
  int s[4][5][99];
  cout << test3d(p,4,5,10);
  cout << test3d(q,100,5,10);
  //cout << test3d(r,4,99,10);
  //cout << test3d(s,4,5,99);
}
```

- ต้องระบุขนาดของทุกระดับชั้นที่ไม่ใช่นอกสุด
- ชั้นนอกสุดเปลี่ยนแปลงขนาดได้
  - ชั้นอื่น ๆ เปลี่ยนไม่ได้

กำหนดค่าเริ่มต้นแบบหลายมิติ

- ใช้ { } ซ้อนกันไปเรื่อย ๆ ได้
- มิติแรกสุดไม่ต้องระบุขนาดก็ได้

```
void print2d(int a[][3], int n, int m) {
  for (int i = 0;i < n;i++) {
    for (int j = 0;j < m;j++)
      cout << a[i][j] << " ";
    cout << endl;
  }
  cout << endl;
}

int main() {
  int a[2][3] = { {1,2,3}, {4,5,6} };
  int b[][2] = { {1,2}, {3,4}, {5,6} };
  int c[][3] = {
    {1,2,3},
    {11,22,33},
    {99,88,55},
    {0,0,0}
  };
  print2d(a,5,3);
  //print2d(b,2,6);
  print2d(c,4,3);

  int p[3][2][3] = {
    { {0,0,0}, {9,9,9} },
    { {1,2,3}, {1,2,3} },
    { {4,5,6}, {6,4,5} },
  };
}
```

อาเรย์แบบระบุขนาดขณะทำงาน

- ตัวอย่างข้างต้นจะเห็นว่ามีการกำหนดขนาดของอาเรย์ไว้เป็นค่าคงที่ตลอดเวลา
- เราสามารถระบุขนาดอาเรย์ให้แปรตามค่าของตัวแปรได้
  - มีข้อจำกัดแปลก ๆ เยอะแยะ ไม่แนะนำให้ใช้ในกรณีอาเรย์ซ้อนกันหลายชั้น

```
void test(int a[],int n) {
  cout << a[n-1] << endl;
}

int main() {
  int n;
  cout << "Enter length: "; cin >> n;
  int v[n] = {0};
  cout << "Size = " << sizeof(v) << endl;
  test(v,10);
}
```

```
//void test2(int n, int m, int a[n][m]) {
//}
int main() {
  int n;
  int m;
  cout << "Enter row: "; cin >> n;
  cout << "Enter col: "; cin >> m;
  int v2[n][m];
  for (int i= 0;i < n;i++)
    for (int j = 0;j < m;j++)
      v2[i][j] = -1;
  //test2(v2,m);
}
```

อาเรย์แบบระบุขนาดขณะทำงาน

- ตามมาตรฐานของ C++ นั้น ตอนประกาศอาเรย์ต้องระบุขนาดเป็นค่าคงที่
- การระบุขนาดโดยใช้ตัวแปรที่เปลี่ยนค่าไปมา จริง ๆ แล้วไม่ใช่มาตรฐานของภาษา C++
- แต่ compiler ของภาษา C++ หลายตัวอนุญาตให้ใช้ได้
- สะดวกเมื่อใช้งานอาเรย์ 1 มิติ
  - ในหลาย ๆ มิติใช้ลำบาก (แต่ก็ใช้ได้)
- ไม่ค่อยแนะนำในวิชานี้

สรุปข้อควรระวังของอาเรย์

- เรียกใช้ [x] ต้องระวังว่า x น้อยกว่าขนาดของอาเรย์
- การหาขนาดของอาเรย์นั้นลำบาก
  - ต้องใช้ sizeof (แถมใช้ในฟังก์ชันไม่ได้)
- ใช้ร่วมกับฟังก์ชันลำบาก ๆ หน่อย
  - ต้องส่งขนาดไปด้วย (เพราะใช้ sizeof ไม่ได้)
  - แก้ไขค่าในฟังก์ชัน ทำให้อาเรย์เปลี่ยนค่านอกฟังก์ชันด้วย
  - หลายมิติ ยิ่งลำบาก
- แนะนำให้กำหนดขนาดเป็นค่าคงตัว

ในบทต่อ ๆ ไปจะได้รู้จัก std::vector ซึ่งเหมาะสมในการใช้งานเป็นอาเรย์ที่มีขนาดแตกต่างกัน มากกว่าการประกาศอาเรย์ตรง ๆ

ตัวอย่างการใช้งานที่พบบ่อย ๆ

```cpp
#include <iostream>
using namespace std;
const int MAX_ROW = 100;
const int MAX_COL = 50;

int main() {
  int matrix[MAX_ROW][MAX_COL];
  int n;
  int m;
  cout << "Enter row: "; cin >> n;
  cout << "Enter col: "; cin >> m;
  if (n > MAX_ROW || m > MAX_COL) {
    cout << "Size too large" << endl;
    return 0;
  }
  for (int i= 0;i < n;i++) {
    cout << "Enter " << m << " values of row " << i << ": " << endl;
    for (int j = 0;j < m;j++) {
      cin >> matrix[i][j];
    }
  }
}
```

- หาค่ามากสุด (น้อยสุด) ในอาเรย์
- แสดงค่าในอาเรย์
- รับข้อมูลเข้าอาเรย์ 2 มิติ (เช่นใช้เก็บข้อมูล matrix)
  - แนะนำให้กำหนดขนาดอาเรย์ให้เพียงพอต่อการรับ