261102
Computer Programming

Lecture 17: Pointers II

Pointer to Array

```
x[5] = { }
cout << x ;
Output : address x , address x[0]
```

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

int main()
{
    int x[] = {55,43,-12};
    int *p = x;
    // or    int *p = &x[0]; = x

    cout << "&x = " << &x << "\n";
    cout << "&x[0] = " << &x[0] << "\n";
    cout << "&x[1] = " << &x[1] << "\n";
    cout << "&x[2] = " << &x[2] << "\n";
    cout << "p = " << p << "\n";
    cout << "p+1 = " << p+1 << "\n";
    cout << "p+2 = " << p+2 << "\n";
    return 0;
}
```

Output

```
&x = 0x22fe20
&x[0] = 0x22fe20
&x[1] = 0x22fe24
&x[2] = 0x22fe28
p = 0x22fe20
p+1 = 0x22fe24
p+2 = 0x22fe28
```

+4

+8

- Array name already pointer
- Array name **x** is a pointer to **&x**[0]
- Return data type of **&x** is **int (*) [3]**
- Return data type of **&x[0]** is **int ***

Pointer to Array

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

int main()
{
    double x[] = {55,43,-12};
    double *p = x;
    // or    double *p = &x[0];

    cout << "&x = " << &x << "\n";
    cout << "&x[0] = " << &x[0] << "\n";
    cout << "&x[1] = " << &x[1] << "\n";
    cout << "&x[2] = " << &x[2] << "\n";
    cout << "p = " << p << "\n";
    cout << "p+1 = " << p+1 << "\n";
    cout << "p+2 = " << p+2 << "\n";
    return 0;
}
```

Output

```
&x = 0x22fe20
&x[0] = 0x22fe20
&x[1] = 0x22fe28
&x[2] = 0x22fe30
p = 0x22fe20
p+1 = 0x22fe28
p+2 = 0x22fe30
```

p is pointer to **double**

Now address shift (+1) = 8 bytes

Pointer to Array

"ตัวชี้ไปที่ array"

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int x[] = {55,43,-12};
7       int (*p) [3] = &x;  ≠ x
8
9       cout << "p = " << p << "\n";
10      cout << "p+1 = " << p+1 << "\n";
11      cout << "p+2 = " << p+2 << "\n";
12      return 0;
13  }
```

Output

```
p = 0x22fe20
p+1 = 0x22fe2c
p+2 = 0x22fe38
```

+12 +24

p is pointer to 3-element int array

Now address shift (+1) = 12 bytes

(4*3)

cout << char *

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

int main()
{
    char x[] = "Strong";
    cout << "&x = " << &x << "\n";
    cout << "&x[0] = " << &x[0] << "\n";
    cout << "&x[1] = " << &x[1] << "\n";
    cout << "&x[2] = " << &x[2] << "\n";
    cout << "&x[3] = " << &x[3] << "\n";
    cout << "&x[4] = " << &x[4] << "\n";
    cout << "&x[5] = " << &x[5] << "\n";
    return 0;
}
```

Output

```
&x = 0x22fe40
&x[0] = Strong
&x[1] = trong
&x[2] = rong
&x[3] = ong
&x[4] = ng
&x[5] = g
```

ปกติ cout ค่า pointer ของ type อื่น
จะแสดง address อันนี้แม่ง Indy

char array is a special case that have different behavior on `cout << char *`

cout << char *

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       char x[] = "Strong";
7       cout << "&x = " << (int *) &x << "\n";
8       cout << "&x[0] = " << (int *) &x[0] << "\n";
9       cout << "&x[1] = " << (int *) &x[1] << "\n";
10      cout << "&x[2] = " << (int *) &x[2] << "\n";
11      cout << "&x[3] = " << (int *) &x[3] << "\n";
12      cout << "&x[4] = " << (int *) &x[4] << "\n";
13      cout << "&x[5] = " << (int *) &x[5] << "\n";
14      return 0;
15  }
```

Output

```
&x = 0x22fe40
&x[0] = 0x22fe40
&x[1] = 0x22fe41
&x[2] = 0x22fe42
&x[3] = 0x22fe43
&x[4] = 0x22fe44
&x[5] = 0x22fe45
```

ถ้าอยากจะโชว์เป็น Address

Using type casting to other pointer types such as **int *** (pointer to int), **void *** (void pointer)

- **&x** returns **char (*) [7]** (pointer to char array of 7-element)
- **&x[0]** returns **char (*)** (pointer to char)

Pointers to Array Declarations

This pointer points to N-element array of this type

(incremental step size is size of N-element array)

type (* name) [N]

This variable type is pointer

Name of Pointer Variable

Pointers to Array Declarations

```
int x[] = {1,2,3};

int * a = x;
int * b = &x[0];

int (*c) [3] = &x;
```

Point to first element (int)

Point to whole array (int[3])

Pointer Arithmetic

- Increment/decrement pointer (++ or --)
- Add/subtract an integer to/from a pointer (+ or += , - or -=)
- Pointers may be subtracted from each other
- Pointer arithmetic meaningless unless performed on pointer to array

Pointer Arithmetic

```
int x = 55;
double y = 55.55;
int *p = &x;
cout << "&x =" << &x << "\n";
cout << "p =" << p << "\n";
cout << "p+1 =" << p+1 << "\n";

double *q = &y;
cout << "&y =" << &y << "\n";
cout << "q =" << q << "\n";
cout << "q+1 =" << q+1 << "\n";
```

Output

```
&x =0x22fe2c
p =0x22fe2c +4
p+1 =0x22fe30
&y =0x22fe20
q =0x22fe20 +8
q+1 =0x22fe28
```

Pointer Arithmetic

```
int x = 55;
double y = 55.55;
void *p = &x;
cout << "&x =" << &x << "\n";
cout << "p =" << p << "\n";
cout << "p+1 =" << p+1 << "\n";

p = &y;
cout << "&y =" << &y << "\n";
cout << "p =" << p << "\n";
cout << "p+1 =" << p+1 << "\n";
```

Output

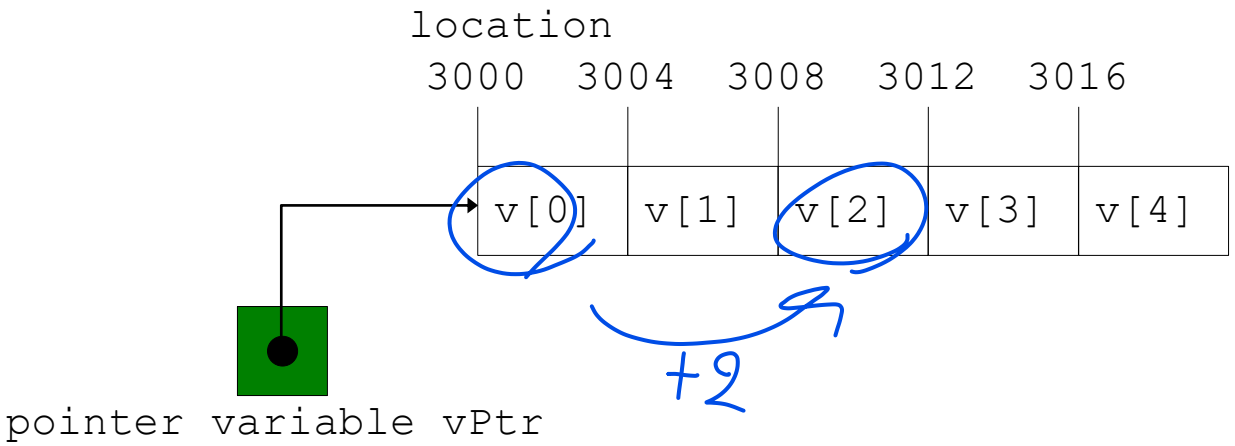
```
&x =0x22fe34
p =0x22fe34 +1
p+1 =0x22fe35
&y =0x22fe28
p =0x22fe28 +1
p+1 =0x22fe29
```

Pointer Arithmetic

- 5 element **`int`** array on a machine using 4 byte **`int`**s
  - **`vPtr`** // points to first element **`v[0]`**

    `vPtr = 3000`
  - **`vPtr += 2`**; // sets **`vPtr`** to **`3008`**

    `vPtr` points to **`v[ 2 ]`**



Pointer Arithmetic

- Subtracting pointers
  - Returns number of elements between two addresses

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

int main()
{
    int x[] = {55,43,-12};
    int *p1 = &x[0];
    int *p2 = &x[2];

    cout << "p1 = " << p1 << "\n";
    cout << "p2 = " << p2 << "\n";
    cout << "p2-p1 = " << p2-p1 << "\n";
    return 0;
}
```

Output

```
p1 = 0x22fe20
p2 = 0x22fe28
p2-p1 = 2
```

จำนวน element ที่ห่างกัน

Pointer Comparison

- Use equality and relational operators
- Comparisons meaningless unless pointers point to members of same array
- Compare addresses stored in pointers
- Example: could show that one pointer points to higher numbered element of array than other pointer
- Common use to determine whether pointer is 0 (does not point to anything: NULL)

Pointer Comparison

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

int main()
{
    int x[] = {1,2,3};
    int *p1 = &x[0];
    int *p2 = &x[2];

    if(p2 > p1) cout << "p2 is point to the element after p1.\n";
    else if(p2 < p1) cout << "p2 is point to the element before p1.\n";
    else cout << "p2 is point to the same element at p1.\n";

    if(p2 == 0) cout << "p2 is a NULL pointer.\n";
    else cout << "p2 is not a NULL pointer.\n";

    return 0;
}
```

Output

```
p2 is point to the element after p1.
p2 is not a NULL pointer.
```

Dereferencing Operator

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int x[] = {55,43,-12};
7       int *p = x;
8
9       cout << "p = " << p << "\n";
10      cout << "*p = " << *p << "\n";
11      cout << "*(p+1) = " << *(p+1)<< "\n";
12      cout << "*(p+2) = " << *(p+2)<< "\n";
13      cout << "*p+1 = " << *p+1 << "\n";
14      return 0;
15  }
```

Output

```
p = 0x22fe20
*p = 55
*(p+1) = 43
*(p+2) = -12
*p+1 = 56
```

*p	equivalent to	x[0]
*(p+1)	equivalent to	x[1]
*(p+2)	equivalent to	x[2]

Array and Pointer Names

(ใช้แทนกันได้)

- Arrays and pointers closely related
  - Array name like constant pointer
  - Pointers can do array subscripting operations
- Accessing array elements with pointers
  - Element `b[n]` can be accessed by `*(bPtr + n)`
    - Called pointer/offset notation
  - Addresses
    - `&b[3]` same as `bPtr + 3`
  - Value : Array name can be treated as pointer
    - `b[3]` same as `*(b + 3)`
  - Pointers can be subscripted (pointer/subscript notation)
    - `bPtr[3]` same as `b[3]`

Array and Pointer Names

- A pointer variable can be used to access the elements of an array of the same type.

```
int gradeList[8] = {92,85,75,88,79,54,34,96};
int *myGrades = gradeList;

cout << gradeList[0] << " ";
cout << gradeList[2] << " ";

cout << *myGrades << " ";
cout << *(myGrades + 2) << " ";

cout << myGrades[3] << " ";
cout << *gradeList << " ";
cout << *(gradeList + 2);
```

Array-style

Pointer-style

Use pointer name as array name

Use array name as pointer name

Output

92	75	92	75	88	92	75

Note that the array name **`gradeList`** acts like the pointer variable **`myGrades`**.

Pointers to 2D Arrays

score[0] score[1]

```
int score[2][3] = {{92,85,75},{88,79,54}};
int *sPtr = score;
```

score[0][0]

[Error] cannot convert 'int (*)[3]' to 'int*' in initialization

`int *sPtr = score;` is equivalent to `int *sPtr = &score[0];`

`score[0]` is int array with 3 elements = {92,85,75}

`&score[0]` returns `int (*) [3]` not `int *`

```
int score[2][3] = {{92,85,75},{88,79,54}};
int (*sPtr) [3] = score;
```

In this case, sPtr++ will move the address by 4*3 bytes

Pointers to 2D Arrays

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

int main()
{
    int score[2][3] = {{92,85,75},{88,79,54}};
    int (*sPtr) [3] = score;

    cout << "sPtr = " << sPtr << "\n";
    cout << "sPtr+1 = " << sPtr+1 << "\n";
    cout << "*sPtr = " << *sPtr << "\n";
    cout << "**sPtr = " << **sPtr << "\n";
    cout << "*(sPtr+1) = " << *(sPtr+1) << "\n";
    cout << "**(sPtr+1) = " << **(sPtr+1) << "\n";

    return 0;
}
```

sPtr →	92	85	75
sPtr+1 →	88	79	54

sPtr = &score[0]

* sPtr = ~~*&~~score[0]

= score[0]

= &score[0][0] int *

*sPtr →	92	85	75
	88	79	54

Output

```
sPtr = 0x22fe20
sPtr+1 = 0x22fe2c
*sPtr = 0x22fe20
**sPtr = 92
*(sPtr+1) = 0x22fe2c
**(sPtr+1) = 88
```

Pointers to 2D Arrays

int [2][3]

score

acts as int **

int [3]

score[0]

acts as int *

score[1]

int

score[0][0]

score[0][1]

score[0][2]

score[1][0]

score[1][1]

score[1][2]

Array name acts like a pointer

Pointers to 2D Arrays

sPtr = score; or sPtr = &score[0];

sPtr → score[0] → score[0][0]

Array name of int [3]

int

score[0] is array name that acts as pointer to score[0][0]

Value of score[0] becomes address of score[0][0]

*sPtr = address of score[0][0]

Pointers to 2D Arrays

sPtr → score[0] → score[0][0]

Array name of int [3]

int

**sPtr = *score[0] = value of score[0][0]

score[0] is array name that acts as pointer to score[0][0]

---

sPtr+1 → score[1] → score[1][0]

Array name of int [3]

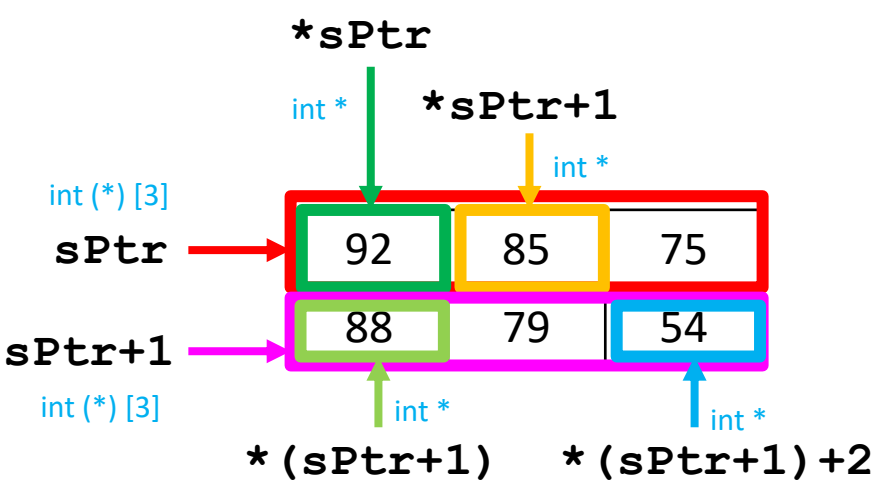
int

(Act as pointer stores address of score[1][0])

Pointers to 2D Arrays

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

int main()
{
    int score[2][3] = {{92,85,75},{88,79,54}};
    int (*sPtr) [3] = score;

    cout << "**sPtr = " << **sPtr << "\n"; //score[0][0]
    cout << "**(sPtr+1) = " << **(sPtr+1) << "\n"; //score[1][0]
    cout << "*(*sPtr+1) = " << *(*sPtr+1) << "\n"; //score[0][1]
    cout << "*(*(sPtr+1)+2) = " << *(*(sPtr+1)+2) << "\n";  //score[1][2]

    return 0;
}
```

sPtr is int (*) [3]
*sPtr is int *
**sPtr is int



Output

```
**sPtr = 92
**(sPtr+1) = 88
*(*sPtr+1) = 85
*(*(sPtr+1)+2) = 54
```

92

88

Pointers to 2D Arrays

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

int main()
{
    int score[2][3] = {{92,85,75},{88,79,54}};
    int (*sPtr) [3] = score;

    cout << "sPtr[0][0] = " << sPtr[0][0] << "\n"; //score[0][0]
    cout << "*sPtr[1] = " << *sPtr[1] << "\n"; //score[1][0]
    cout << "(*sPtr)[1] = " << (*sPtr)[1] << "\n"; //score[0][1]

    cout << "**score = " <<  **score << "\n";  //score[0][0]
    cout << "**(score+1) = " <<  **(score+1) << "\n";  //score[1][0]
    cout << "*(*score+1) = " <<  *(*score+1) << "\n";  //score[0][1]

    return 0;
}
```

Use pointer name as array name

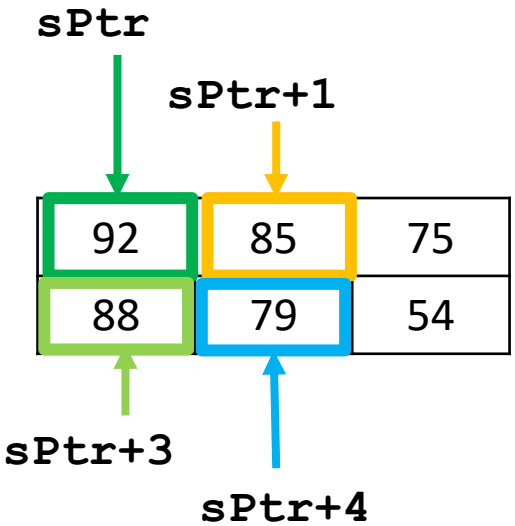
Use array name as pointer name

Output

```
sPtr[0][0] = 92
*sPtr[1] = 88
(*sPtr)[1] = 85
**score = 92
**(score+1) = 88
*(*score+1) = 85
```

Pointers to 2D Arrays

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int score[2][3] = {{92,85,75},{88,79,54}};
7       int *sPtr = score[0];
8       // or int *sPtr = &score[0][0];
9
10      cout << "sPtr = " << sPtr << "\n";
11      cout << "sPtr+1 = " << sPtr+1 << "\n";
12      cout << "*sPtr = " << *sPtr << "\n";
13      cout << "*(sPtr+1) = " << *(sPtr+1) << "\n";
14      cout << "*(sPtr+4) = " << *(sPtr+4) << "\n";
15
16
17      return 0;
18  }
```



Output

```
sPtr = 0x22fe20
sPtr+1 = 0x22fe24
*sPtr = 92
*(sPtr+1) = 85
*(sPtr+4) = 79
```

Array values are stored in consecutive addresses.

Arrays of Pointers

Array that contains N pointers

```
type *name[N]
```

Pointer that points to array

```
type (* name) [N]
```

Arrays of Pointers

```
int x[] = {1,2,3};

int * a = x;
int * b = &x[0];

int (*c) [3] = &x;

int * d [3];
int * (e[3]);
```

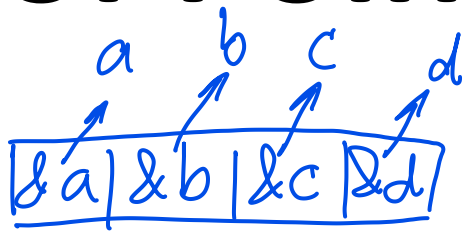
Point to first element (int)

Point to whole array (int[3])

[int* int* int*]

Array of three pointers
(point to int)

Arrays of Pointers



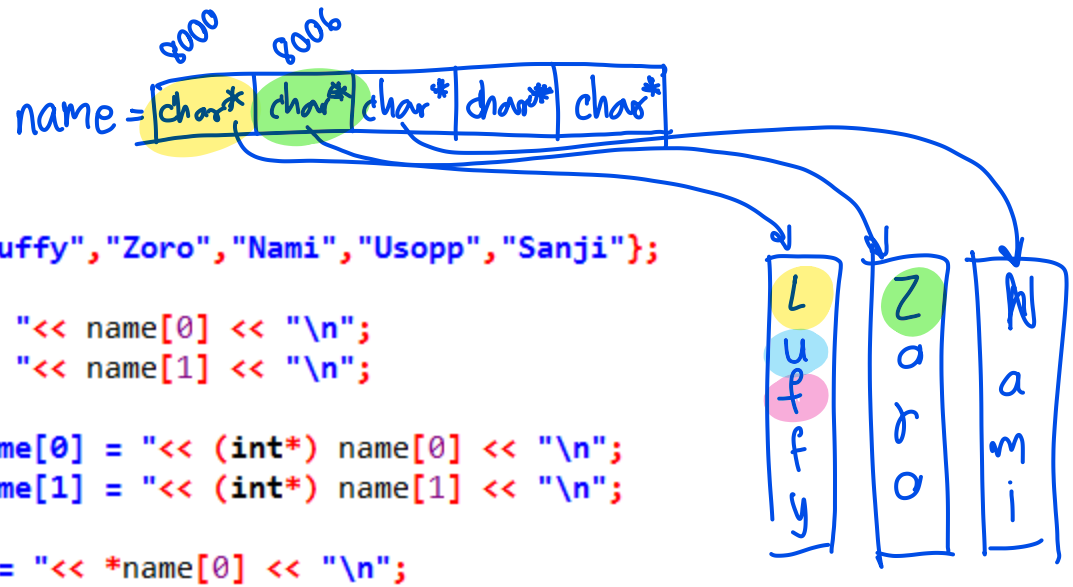

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  using namespace std;
5
6  int main()
7  {
8      int a = 50, b = 100, c = 500, d = 1000;
9      int *sPtr[4];
10     sPtr[0] = &a; sPtr[1] = &b; sPtr[2] = &c; sPtr[3] = &d;
11
12     cout << "sPtr[0] = " << sPtr[0] << "\n";
13     cout << "sPtr[1] = " << sPtr[1] << "\n";
14     cout << "sPtr[2] = " << sPtr[2] << "\n";
15     cout << "sPtr[3] = " << sPtr[3] << "\n";
16
17     int temp,idx1,idx2;
18     srand(time(0));
19     for(int i = 0; i < 69; i++){
20         idx1 = rand()%4;  0
21         idx2 = rand()%4;  2
22         temp = *sPtr[idx1];  a
23         *sPtr[idx1] = *sPtr[idx2];  c
24         *sPtr[idx2] = temp;
25     }
26
27     cout << "a = " << a << "\n";
28     cout << "b = " << b << "\n";
29     cout << "c = " << c << "\n";
30     cout << "d = " << d << "\n";
31
32     return 0;
33 }
```

Output

```
sPtr[0] = 0x22fe2c
sPtr[1] = 0x22fe28
sPtr[2] = 0x22fe24
sPtr[3] = 0x22fe20
a = 1000
b = 500
c = 100
d = 50
```

Arrays of Pointers

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

int main()
{
    char *name[] = {"Luffy","Zoro","Nami","Usopp","Sanji"};

    cout << "name[0] = "<< name[0] << "\n";
    cout << "name[1] = "<< name[1] << "\n";

    cout << "(int*) name[0] = "<< (int*) name[0] << "\n";
    cout << "(int*) name[1] = "<< (int*) name[1] << "\n";

    cout << "*name[0] = "<< *name[0] << "\n";
    cout << "*name[1] = "<< *name[1] << "\n";

    cout << "*(name[0]+1) = "<< *(name[0]+1) << "\n";
    cout << "*(name[0]+2) = "<< *(name[0]+2) << "\n";


    return 0;
}
```



Output

```
name[0] = Luffy
name[1] = Zoro
(int*) name[0] = 0x488000
(int*) name[1] = 0x488006
*name[0] = L
*name[1] = Z
*(name[0]+1) = u
*(name[0]+2) = f
```

Similar to

```
string name[] = {"Luffy","Zoro","Nami","Usopp","Sanji"};
cout << "name[0] = "<< name[0] << "\n";
cout << "name[1] = "<< name[1] << "\n";
// Deferencing operator * can not be used for C++ string object
```

Example 17-A: Destiny Draw

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

void deckShuffle(int *);
void showDrawSequence(int [][13]);
void draw(int [][13], int *);
char suit[] = "\3\4\5\6";
char *face[] = {"A","2","3","4","5","6","7","8","9","10","J","Q","K"};

int main()
{
    int deck[4][13] = {};
    int count = 1;
    srand(time(0));
    deckShuffle(&deck[0][0]);
    showDrawSequence(deck);

    for(int i = 0; i < 3;i++){
        draw(deck,&count);
    }

    return 0;
}
```

Array of char

Array of char *

```
------------- Draw Sequence -------------
     A  2  3  4  5  6  7  8  9 10  J  Q  K
♥   22 35 20  6 36 27 41 21 32 15 43 31 52
♦   16 18 26 50 37  4 24 30 33 23 47 44 11
♣   39 34  7 13 45 14  8 49 12 40  5 25  1
♠    2 51 10 29 48 19 46 17 28 42  9 38  3
-----------------------------------------
You got K♣
You got A♠
You got K♠
```

Example 17-A: Destiny Draw

Pass by pointer (point to `deck[0][0]`)

```
void deckShuffle(int *dptr){
    int row,col;
    for(int i = 1; i <= 52; i++){
        do{
            row = rand()%4;
            col = rand()%13;
        }while(*(dptr+col+13*row) != 0);
        *(dptr+col+13*row) = i;
    }
}
```

If `deck[row][col]` is equal to 0
set `deck[row][col] = i`

```
deckShuffle(&deck[0][0]);
```

Function calling (pass address of `deck[0][0]`)
pointer `dptr = &deck[0][0]`

dptr → `deck[0][0]`
dptr+1 → `deck[0][1]`
dptr+2 → `deck[0][2]`

dptr+12 → `deck[0][12]`
dptr+13 → `deck[1][0]`
dptr+13+1 → `dect[1][1]`

Example 17-A: Destiny Draw

```
38 void showDrawSequence(int d[][13]){
39     cout << "------------- Draw Sequence -------------\n";
40
41     cout << "    ";
42     for(int j = 0; j < 13; j++){
43         if(face[j] == "9") cout << face[j] << " ";
44         else cout << face[j] << "  ";
45     }
46     cout << "\n";
47
48     for(int i = 0;i < 4; i++){
49         cout << suit[i] << "  ";
50         for(int j = 0; j < 13; j++){
51             if(d[i][j] <= 9) cout << " ";
52             cout << d[i][j] << " ";
53         }
54         cout << "\n";
55     }
56     cout << "----------------------------------------\n";
57 }
```

Function call

```
18     showDrawSequence(deck);
```

```
-------------- Draw Sequence --------------
    A  2  3  4  5  6  7  8  9 10  J  Q  K
♥  22 35 20  6 36 27 41 21 32 15 43 31 52
♦  16 18 26 50 37  4 24 30 33 23 47 44 11
♣  39 34  7 13 45 14  8 49 12 40  5 25  1
♠   2 51 10 29 48 19 46 17 28 42  9 38  3
-------------------------------------------
You got K♣
You got A♠
You got K♠
```

Example 17-A: Destiny Draw

Pass by array (deck)

Pass by pointer (point to count)

```
void draw(int d[][13], int *c){
    for(int i = 0;i < 4; i++){
        for(int j = 0; j < 13; j++){
            if(d[i][j] == *c){
                cout << "You got " << face[j] << suit[i] << "\n";
                (*c)++;
                return;
            }
        }
    }
}
```

char *

char