Arrays, Pointers and Struct

ICT106 Fundamentals of Computer Systems
Eric Li

Objectives

- Provide practical examples for using arrays and pointers
- Provide an introduction to pointer arithmetic operations
- Further reading:
  Chapter 10, *C Primer Plus*, 5th ed
  Chapter 5, 12, 13, *Practical C Programming*, 3rd ed

Arrays and Pointers in C

- Array and pointer are sometimes interchangeable;
- Advantages of array
  ■ More convenient;
  ■ Simpler;
  ■ Safer;
- Advantages of pointer
  ■ More flexible;
  ■ More dynamic;

```c
int results[5] = {1, 4, 5, -8, 10};
int *ipr;
ipr = &results[0];
ipr = &results[4];
ipr = a;  *ipr = 2;
```

results:

<table>
<thead>
<tr>
<th></th>
<th>0x10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>-8</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

pr:

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
</tr>
<tr>
<td>?</td>
</tr>
<tr>
<td>?</td>
</tr>
</tbody>
</table>


int results[5] = {1, 4, 5, -8, 10};
int *ipr;
ipr = &results[0];
ipr = &results[4];
ipr = a;
*ipr = 2;

int results[5] = {1, 4, 5, -8, 10};
int *ipr;
ipr = &results[0];
ipr = &results[4];
ipr = a;
*ipr = 2;

int results[5] = {1, 4, 5, -8, 10};
int *ipr;
ipr = &results[0];
ipr = &results[4];
ipr = a;
*ipr = 2;

int results[5] = {1, 4, 5, -8, 10};
int *ipr;
ipr = &results[0];
ipr = &results[4];
ipr = a;
*ipr = 2;
int results[5] =
    {1, 4, 5, -8, 10};
int *ipr;
ipr = &results[0];
ipr = &results[4];
ipr = a;
*ipr = 2;

Using pointer in array manipulation

for(int i=0; i<5; i++) {
ipr = ipr + 1;
    *ipr = *ipr * i;
}

- Side effect: (Be careful)
  *(ipr++) *= i;

Passing by value

main() {
    int num = 4;
    int result = sqr(num);
}
int sqr(int out) {
    out = out * out;
    return out;
}

Passing by value

main() {
    int num = 4;
    int result = sqr(num);
}
int sqr(int out) {
    out = out * out;
    return out;
}
Passing by value

```c
main() {
    int num = 4;
    int result = sqr(num);
}
int sqr(int out) {
    out = out * out;
    return out;
}
```

Passing by reference

```c
main() {
    int num = 4;
    sqr(&num);
}
void sqr(int *prt) {
    *prt = (*prt) * (*prt);
}
```
Passing by reference

```c
main() {
    int num = 4;
    sqr(&num);
}
void sqr(int *prt) {
    *prt = (*prt) * (*prt);
}
```

Passing by reference: example

```c
main() {
    int num[5];
    getInput(num, 5);
}
void getInput(int in[], int len) {
    int i=0;
    for(; i<len; i++) {
        scanf("%d", &in[i]);
    }
}
```

Passing by pointer: example

```c
main() {
    int num[5];
    getInput(num, 5);
}
```

`const` Keyword

```c
int main()
{
    int num = 7;
    int out = sqr(&num);
    printf("Square of %d is %d\n", num, out);
}
int sqr(const int *pt)
{
    return (*pt) * (*pt);
}
```
What if …

```c
int sqr(const int *pt)
{
    *pt = 9;
    return ((*pt) * (*pt));
}
```

Multi-dimensional array (option)

- Recall: Multi-dimensional arrays are represented in memory as a plain one dimensional array.

```c
double array[2][4] = {{1, 4, 5, 3}, {9, 0, 8, 2}};
double *pr = &array[0][0];
for(i=0; i<2; i++) {
    for(j=0; j<4; j++) {
        printf("%lf ", *(pr+(i*4)+j));
    }
    printf("\n");
}
```

Pointer Arithmetic

- Pointers can be operated just like other data type
- Basic operations include:
  - Assignment;
  - Dereferencing;
  - Adding and Subtracting;
  - Differencing;
  - Comparison;
  - (Option) Indirect pointers

Assignment and Dereferencing

- Value of a pointer can be assigned to another pointer

```c
char a = ‘V’, b = ‘4’;
char *apt, *bpt;
apt = &a;
bpt = apt;
b = *bpt; //dereferencing
```
Assignment and Dereferencing

- Value of a pointer can be assigned to another pointer

```c
char a = 'V', b = '4';
char *apt, *bpt;
apt = &a;
bpt = apt;
b = *bpt; //dereferencing
```

Assignment and Dereferencing

- Value of a pointer can be assigned to another pointer

```c
char a = 'V', b = '4';
char *apt, *bpt;
apt = &a;
bpt = apt;
b = *bpt; //dereferencing
```

Assignment and Dereferencing

- Value of a pointer can be assigned to another pointer

```c
char a = 'V', b = '4';
char *apt, *bpt;
apt = &a;
bpt = apt;
b = *bpt; //dereferencing
```

Assignment and Dereferencing

- Value of a pointer can be assigned to another pointer

```c
char a = 'V', b = '4';
char *apt, *bpt;
apt = &a;
bpt = apt;
b = *bpt; //dereferencing
```

Pointer Arithmetic Operations

- Change the content of a pointer effectively redirects the pointer to a new address;
- Because a pointer can reference to virtually ANY memory location, use pointer operations with cautions.
**NOTE**

- NEVER dereference an uninitialised pointer;
- ALWAYS add or subtract a pointer with an INTEGER value;
- Always check the valid range of an array;
- Use `const` keyword to protect change of variables;
- Use Array type when possible;
- DO NOT use indirect pointers unless you know what you are doing.

---

**Adding, Subtracting and Differencing**

- Make the pointer move to the next element of the array, e.g.

  ```
  int *ipt, *iipt;
  double *dpt;
  *(ipt + 3); // increase by six bytes
  *(dpt + 3); // increase by 12 bytes
  ```

- Note that the incremental values of the last two operations are not the same.
- Note the difference between (*ipt + 3) and *(ipt + 3)

---

**Differencing**

- Differencing gives the distance of two elements stored in an array

  ```
  int array[] = {3, 6, 70, 2, 10};
  int *ipt = array;
  iipt = array + 3;
  iipt – ipt; // Gives 3 ints
  ```

---

**Comparison**

- Use relational operations to compare the values of two pointers, e.g.

  ```
  int *pr, *apr, *bpr;
  bpr = pr;
  
  if(*pr == *apr) {…} // NOT ( pr == apr)
  while( *apr >= 56 ) {…}
  boolean flag = ( bpr == pr); //EQUIVALENT ( *bpr == *pr)
  ```
Indirect Pointers (option)

- You can create a pointer to store an address of another pointer. Such pointer is called Indirect Pointer.
- Indirect pointers are represented by double asterisks `**` e.g.

```c
int *pt = &var;
int* *ppt;
**ppt = &pt;
```

```plaintext
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ppt</td>
<td>pt</td>
</tr>
<tr>
<td>var</td>
<td>801</td>
</tr>
</tbody>
</table>
```