Structured Programming

- Pointer

Dr Donglong Chen

With Thanks to Dr. Xin Feng and Dr. Haipeng Guo

Outline

- Memory
- Values, variable, pointers
- Pointer, pointee
- Pointer and array
- Pointer and structure

Mail Boxes

Three elements: Box no., mail, owner of mail

Memory

- The computer's memory is a sequential collection of “storage cells”
- Each cell can store one byte
- Each cell has an address
- When a normal variable is assigned a value, the value is put in the cells allocated to this variable

Three elements: value, variable, address

`(mail) (owner) (box no.)`

Memory

```
int m = 1;
```

00000000	00000000	00000000	00000001

0x377a

1 is the value of the variable m. It is put in the cells starting from address 0x377a

The postal card is owned by Judy. It is put in the mailbox 303

Data Types and Storage Bytes

- Values of different types occupy different storage bytes
  - int: 4 bytes
  - char: 1 byte
  - float: 4 bytes
  - double: 8 bytes

Accessing the Address of A Variable

- The operator & can be used to accesses the address of a variable
  - E.g.,
    - `int m;` then `&m` indicates the address of variable `m`
    - `int grade[10];` then `&grade[5]` indicates the address of `grade[5];`
- The elements in an array are stored in consecutive cells. The number of cells that are used to save an element depends on the element type.
- The array name can be directly used as the starting address of an array
  - E.g.
    - `int grade[10]; grade` is the address of the first element, so grade is equivalent to `&grade[0]`.

An Example

```
int m;
int grade[10];

printf("Please input the value of m:");
scanf("%d", &m);

printf("m = %d is stored at address %x \n", m, &m);
printf("grade[2] is stored at address %x \n",  &grade[2]);
printf("grade[0] is stored at address %x \n",  &grade[0]);
printf("the starting address of the array is %x \n", grade);
```

Pointer and Pointee

- A pointer is a variable that points to or references a memory location in which data is stored.
- A pointer variable's value is the memory address of the pointee variable

```
int x =3;
int *p = &x;
```

				Address
00000000	00000000	00000000	00000011	0xbfbc758c
0xbf	0xbc	0x75	0x8c	0xbfbc7590

- p's value is x's address.
- p is also allocated cells to store its value
- x is the pointee
- p is the pointer, p points to x

Declare A Pointer

- Format
  - `type *pointername;`

```
int x;
int *p1;
char *p2;

p1 = &x;      /* store the address in p1 */
scanf("%d", p1);  /* i.e. scanf("%d",&x); */

p2 = &x; /* incorrect, error will be given */
```

Declare A Pointer

```c
void main(){

int x;
int *p1;

p1 = &x;      /* store the address in p1 */
scanf("%d", p1);  /* i.e. scanf("%d",&x); */

printf("The address of x is %x\n",p1);
printf("The value of x is %d\n",*p1);
printf("The address of x is %x\n",&x);
printf("The value of x is %d\n",x);
```

C:\windows\system32\cmd.exe

```
56
The address of x is 9efa20
The value of x is 56
The address of x is 9efa20
The value of x is 56
请按任意键继续. . .
```

100 %

Initialize A Pointer

- Like other variables, always initialize pointers before using them!!!

```
int main()
{
  int x;
  int *p;

  scanf("%d", p); /*incorrect, p is not initialized */

  p = &x;
  scanf("%d", p); /* correct, p points to x */
}
```

```
int* p = &x;
```

=

```
int *p = &x;
```

=

```
int *p;
p = &x;
```

Exercise #1

```
int main()
{
  int x = 3;
  int *p = &x;

  printf("x = %d\n", x);
  printf("The address of x is %x \n", &x);
  printf("The value of p is %x \n", p);
  printf("The address of p is %x \n", &p);
  return 0;
}
```

Output?

x

00000000	00000000	00000000	00000011
0xbf	0xbc	0x75	0x8c

0xbfbc758c

0xbfbc7590

Dereferencing

- &Pointee: Pointee's address
- *Pointer: value stored in the memory location pointed by the pointer (*Pointer is called dereferencing)

```
int x = 3;
int *p = &x;

printf("x = %d\n", x);
printf("x = %d\n", *p);

*p = 4;
printf("x = %d\n", *p);
printf("x = %d\n", x);
```

Output?

An Example

```
int main()
{
  int x, *p;
  p = &x;        /* initialise pointer */

  *p = 0;        /* set x to zero */
  printf("x is %d\n", x);
  printf("*p is %d\n", *p);

  *p += 1;         /* increment what p points to */
  printf("x is %d\n", x);

  (*p)++;          /* increment what p points to */
  printf("x is %d\n", x);
  return 0;
}
```

Exercise #2

```
int main()
{
  int a = 100, b = 88, c = 8;
  int *p1 = &a, *p2, *p3 = &c;

  p2 = &b;  // p2 points to b
  p2 = p1;  // p2 points to a
  b = *p3;  //assign c to b
  *p2 = *p3;      //assign c to a
  printf("%d %d %d",a,b,c);
}
```

What is the output?

Exercise #3

```
int main()
{
  int value1 = 5, value2 = 15;
  int *p1, *p2;

  p1 = &value1;
  p2 = &value2;

  *p1 = 10;
  *p2 = *p1;

  p1 = p2;
  *p1 = 20;

  printf("%d %d", value1, value2);
}
```

What is the output?

Exercise #4

```
int main()
{
  int a = 3;
  char s  = 'z';
  double d = 1.03;

  int *pa = &a;
  char *ps = &s;
  double *pd = &d;

  printf("%d %d %d\n", sizeof(pa), sizeof(*pa), sizeof(&pa));
  printf("%d %d %d\n", sizeof(ps), sizeof(*ps), sizeof(&ps));
  printf("%d %d %d\n", sizeof(pd), sizeof(*pd), sizeof(&pd));
}
```

What is the output?

Return Values Through Pointers

```
double calSumAverage(double, double, double* );
int main( )
{
  double x = 1.0, y = 2.0;
  double average, sum;
  sum = calSumAverage(x , y, &average );
  printf("The sum is %f, the average is %f", sum, average);
  return 0;
}
double calSumAverage(double no1, double no2, double
   *pAverage)
{
    double sum;
    sum = no1 + no2;
    *pAverage = sum / 2;
    return sum;
}
```

Remember this program?

Compare These Two Cases

The value of average is changed after the function call

```
int main( )
{
   ......
   calSumAverage(&average );
   ......
}
double calSumAverage(double *pAverage)
{
      ......
      *pAverage = sum / 2;
      ......
}
```

Pass the address

Compare These Two Cases

The value of average is NOT changed after the function call

```
int main( )
{
   ......
   calSumAverage(average );
   ......
}
double calSumAverage(double pAverage)
{
      ......
      pAverage = sum / 2;
      ......
}
```

Pass the value

An Example

```
void swap(char *p1, char *p2)
{
    char temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

int main()
{
    char a = 'y';
    char b = 'n';
    swap(&a, &b);
    printf("%c %c", a, b);
    return 0;
}
```

```
swap(char p1, char p2)
{
  char temp = p1;
  p1 = p2;
  p2 = temp;
}
```

```
swap(a, b)
```

If we use these, can values of `a` and `b` be swapped?

Pointer to Pointer

```
int x = 58;
int *p ;
int **q;

p = &x;
q = &p;

printf("%x %x %x\n", &x, p, *q);
printf("%d %d %d\n", x, *p, **q);
```

x	00000000	00000000	00000000	00111010	0xbfbc758c
p	0xbf	0xbc	0x75	0x8c	0xbfbc7590
q	0xbf	0xbc	0x75	0x90	0xbfbc7594

Pointer to Pointer

```
int x = 58;
int *p ;
int **q;

p = &x;
q = &p;

printf("%x %x %x\n", &x, p, *q);
printf("%d %d %d", x, *p, **q);
```

C:\windows\system32\cmd.exe

```
8ffce4 8ffce4 8ffce4
58 58 58请按任意键继续. . .
```

Array and Pointer

- An array name indicates the address of the first element

```
void main()
{
  int a[5] = {2, 4, 6, 8, 22};
  printf("%d %d %d", *a, a[0], *(&a[0]));
}
```

What is the output?

Pointer and Array

- Given a pointer `p`, (`p + n`) points to the address `p + nk` where `k` is size of data type pointed by `p` in the declaration

```
int a[5] = {2, 4, 6, 8, 22};
int *p;
p = &a[0];
```

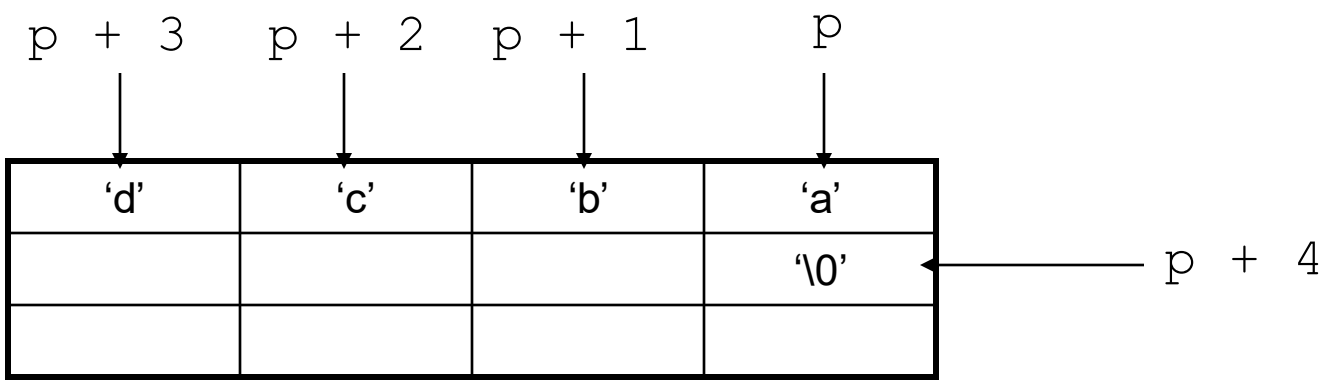
0000000	0000000	0000000	00000010	←	p
0000000	0000000	0000000	00000100	←	p + 1
0000000	0000000	0000000	00000110	←	p + 2
0000000	0000000	0000000	00001000	←	p + 3
0000000	0000000	0000000	00010110	←	p + 4

int a[5]

Pointer and Array

- Given a pointer `p`, (`p + n`) points to the address `p + nk` where `k` is size of data type pointed by `p` in the declaration

```
char a[5] = {'a', 'b', 'c', 'd', '\0'};
char *p;
p = &a[0];
```



Pointer and Array

- `*(p + n)` is same as `a[n]` if `p = &a[0]`
- Since `a = &a[0]`, `*(a + n)` is same as `a[n]` too.

Array of Pointers

- If we have a declaration like

```
int *p[10];
```

that means we have an array, each element in the array is a pointer that points to an int data. E.g.,

```
int a, b, c;
int *p[10];

p[0] = &a;
P[2] = &b;
… …
```

Pointer and String

```
char strA[80] = “Hello";
char strB[80];

char *pA;      /* a pointer to type character */
char *pB;      /* another pointer to type character */

puts(strA);    /* show string A */
pA = strA;     /* point pA at string A */
puts(pA);      /* show what pA is pointing to */
printf(“\n”);   /* move down one line on the screen */

pB = strB;     /* point pB at string B */
while(*pA != '\0')     /* copy strB to strA */
  *pB++ = *pA++;

*pB = '\0';
puts(pB);
puts(strB);            /* show strB on screen */
```

Pointer and Structure

```
typedef struct {
  char forename[20];
  char surname[20];
  float age;
  int childcount;
} person;

person jimmy;
person *p;  // p points to a structure of person type

p = &jimmy; //  initialization, p points to jimmy

p -> age = 30; /* equivalent to jimmy.age = 30 or (*p).age = 30*/

strcpy(p -> surname, “Enns”);
```

Summary

- Pointer uses the address of variables
- Pointer can be used together with any data type, array and structure.