Foundations of C Programming (Structured Programming)

- Pointer(指针)

Outline

- Memory (内存)
- 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” (顺序排列的存储单元)
- Each cell can store one byte(字节)
- Each cell has an address(地址)
- When a normal variable is assigned a value, the value is put in the cells allocated to this variable

Three elements: value, variable, memory address

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

Memory

```
int m = 1; // int type, 4 bytes
```

00000000	00000000	00000000	00000001

`0x377a`
(hex number for an address)

	1	m	0x377a
Three elements:	value,	variable,	memory address
	(mail)	(owner)	(box no.)
	a letter	Judy	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 access 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` has the same value as `&grade[0]` as an address .

An Example

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

printf("Please input the value of m:");
scanf("%d", &m); //read an integer and store it in m's address

 printf("m = %d is stored at address %x \n", m, &m);
 printf("grade[2] is stored at address %x \n",
      &grade[2]);

 //The following two will print the same address
 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 (引用) a memory location in which data is stored.
- A pointer variable‘s value is the memory address of the pointee (被指向的对象) variable, a pointer points to a pointee.

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

Address

0xbfbc758c
(x's address)

0xbfbc7590
(p's address)

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

Declare A Pointer

- Format
  - `type *pointername;`

```
int x;
int *p1; // p1 is a pointer. It points to int type variable
char *p2; // p2 is a pointer. It points to char type variable

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

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

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 */
}
```

Class Exercise

```
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	0xbfbc758c
p	0xbf	0xbc	0x75	0x8c	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; //*p here means p is a pointer

printf("x = %d\n", x);
printf("x = %d\n", *p); //*p means the object pointed by p

//*p here means the value in the memory pointed by p
*p = 4; //*p means the object pointed by p
printf("x = %d\n", *p);
printf("x = %d\n", x);
```

Output?

An Example

```
int main()
{
  int x, *p;
  p = &x;        /* initialize 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;
}
```

Another Example

```
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?

Class Exercise

```
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?

Pointer and Array

- 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

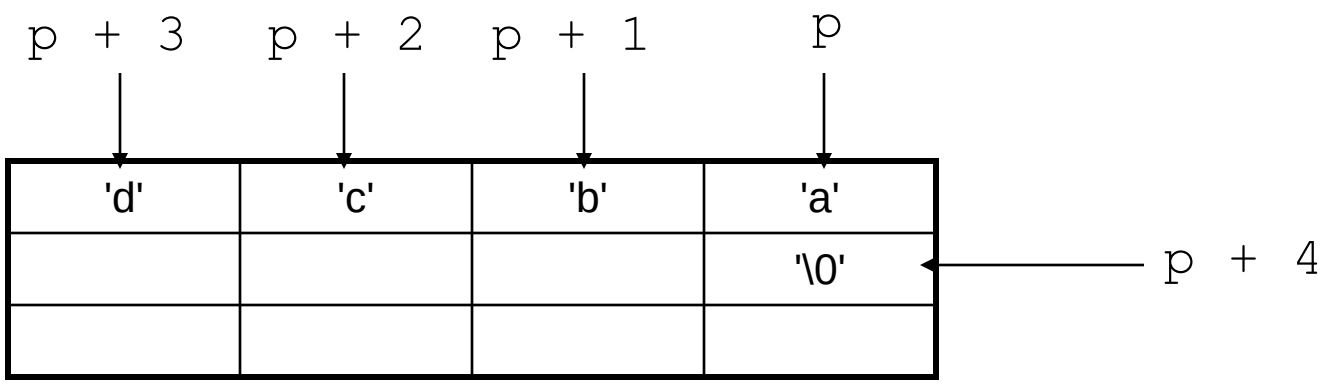
```
int a[5] = {2, 4, 6, 8, 22};
int *p;
p = &a[0];
```

a[5]	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

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

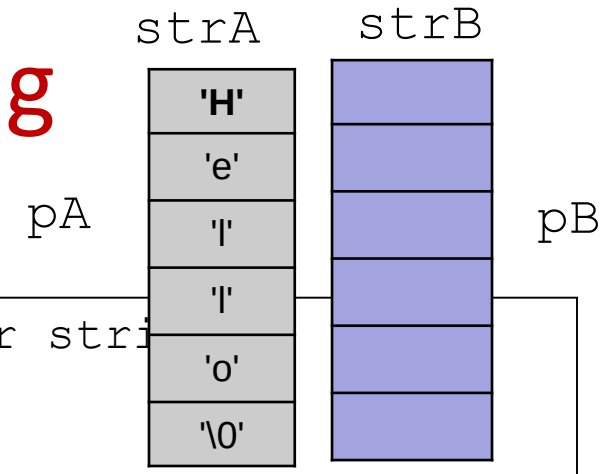
```
int *p[10];
```

that means 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"; //use array for stri
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);      /* pA is same as strA, used as a string*/
printf("\n");    /* move down one line on the screen */

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

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

Pointer and String

- Two ways to declare a string
  - `char str[10]`
  - `char *ps;`
- Both can be used as a string variable
  - `scanf ("%s", str);`
  - `ps = str;`

    `scanf ("%s", ps);`
- The difference is that the pointer must be initialized before it is used.
- Array name str cannot be changed, but pointer ps can be changed, i.e., there should be no str=.....; statement, but there can be ps=....;

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");
```

Pointer and Function

Remember this attention?

Attention: Formal parameter's value change in sub-program will not affect actual parameter (except array).

Can the values of a and b be swapped in this example?

How can we really swap the values?

```
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;
}
```

Pointer and Function

```
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;
}
```

Use pointer!

Pass address to function rather than pass values.

`p1`, `p2` points to `a` and `b` respectively, so the values of `a` and `b` can be swapped.

Class Exercise

```
#include <stdio.h>
void foo(int *j);
int main(){
  int k = 10;
  foo(&k);
  printf("%d", k);
  return 0;

}
void foo(int *j){
  *j = 0;
}
```

What is the output?

Pass More Values Back to Caller

```
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;
}
```

Pass the address

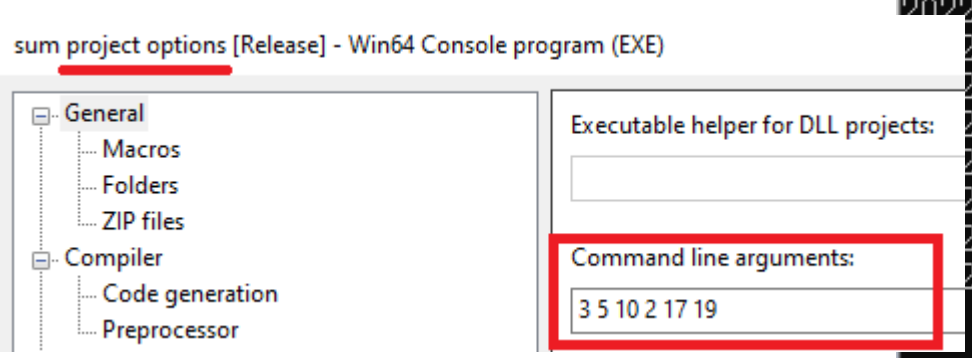
What can this program do?

Arguments(参数) in Main Function

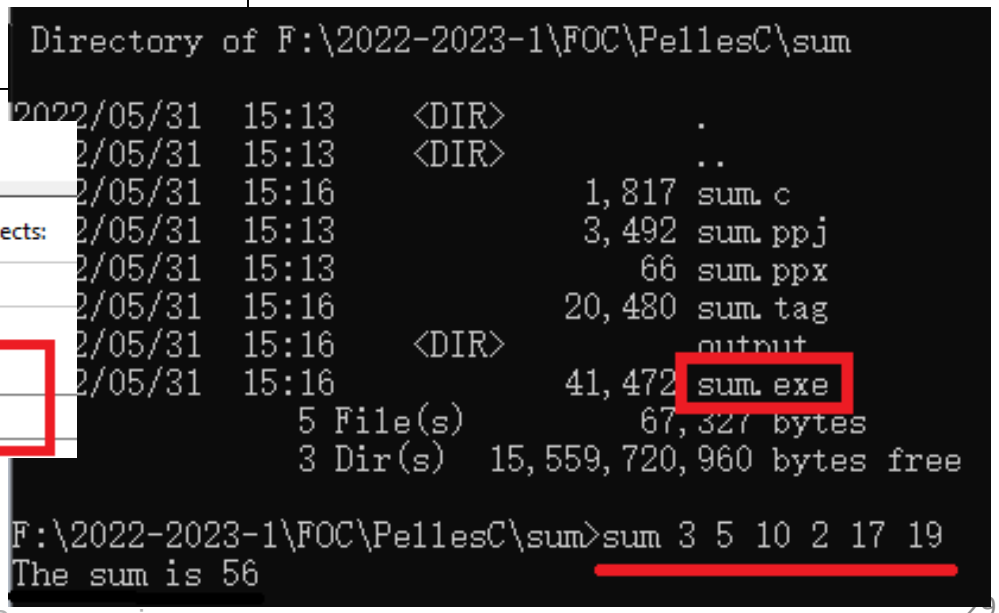
```c
//sum.c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
  int i, sum = 0;
  for (i = 1; i < argc; i++)
      sum = sum + atoi(argv[i]);
  printf("The sum is %d\n", sum);
}
```

```
argc: 7

argv[0]: "sum"
argv[1]: "3"
argv[2]: "5"
....
```



Arguments input in IDE environment



Arguments input in command line interface

Summary

- Pointer uses the address of variables
- Pointer can be used together with any data type, array and structure.
- Pointer to char can be used as a string
- Pointer can be used with functions to let the values changed in function be passed back to main function.