中山大學
SUN YAT-SEN UNIVERSITY

程序设计II实验

电子与信息工程学院（微电子学院）

庞志勇 高级实验师 助教：焦涵博士、桂海田

PPT大纲

- 上次课实验总结
- 本次课的实验介绍

上次课实验总结

- 数组是一种很实用的数据结构，简单易用
- 思考：为何一个很实用的退出功能很多同学做不出来？
- 很多程序设计课程都是在孤立的做“数学题”，抽象没有到具体实用
- 平时读别人的程序要多思考优缺点，如何改进，实用功能改进，算法改进。。。

上次课实验总结

- 前面实验主要培养同学们养成良好的编程习惯（先算法后设计，注释。。。）
- 培养同学们良好的程序设计素养
- 有了良好的编程习惯，学会学好程序设计应该是“水到渠成”
- 学会写实验报告（现实中很多同学到了写毕业论文时，不会写）

本次课的实验介绍

一、实验目的

1.掌握指针的概念和定义方法。

2.掌握指针的操作符和指针的运算。

3.掌握指针与数组的关系。

4.掌握指针与字符串的关系。

5.熟悉指针作为函数的参数及返回指针的函数。

6.了解函数指针。

7. 掌握引用的概念，会定义和使用引用；

8. 掌握函数实参与形参的对应关系，熟悉“地址传递”和“引用传递”的方式。

本次课的实验介绍

二、实验原理

- 内存的问题是软件开发中最核心的问题之一
- 内存的基本和划分单位是字节，每个字节含有8位
- 内存的每个字节都有一个唯一确定的编号，这个编号叫地址
- CPU 只能通过地址来取得内存中的代码和数据，程序在执行过程中会告知 CPU 要执行的代码以及要读写的数据的地址。
- CPU 访问内存时需要的是地址，而不是变量名和函数名！变量名和函数名只是地址的一种助记符，当源文件被编译和链接成可执行程序后，它们都会被替换成地址。
- 编译和链接过程的一项重要任务就是找到这些名称所对应的地址。
- 指针即地址，地址是内存单元的编号，不可重复，但所存内容可以重复。
- 指针变量就是存放内存单元地址的变量。
- 指针能够对地址进行操作使C/C++能够执行更底层的操作。但是如果在程序设计中使用指针出错，可能造成严重的程序异常，甚至会导致死机。
- 指针可以使程序在速度或内存使用方面更有效率，并且可以用来构建复杂的数据结构。如链表、队列、堆栈和树。

由于通过地址能找到所需的变量单元，可以说，地址指向变量单元，打个比方，一个房间的门口挂了一个房间号2008，这个2008就是房间的地址，或者说，2008“指向”该房间。因此，将地址形象化地称为“指针”。意思是通过它能找到以它为地址的内存单元!

本次课的实验介绍

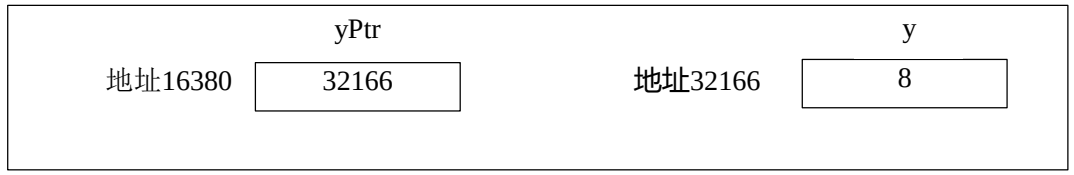
二、实验原理（一图胜千言）

C++ 提供了两种指针运算符:

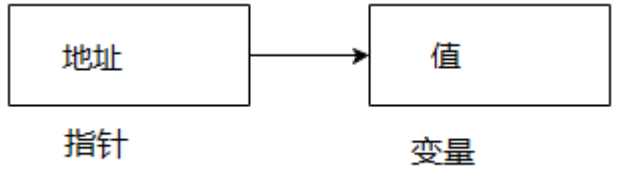
- 一种是取地址运算符 &
- 一种是间接寻址运算符 *。

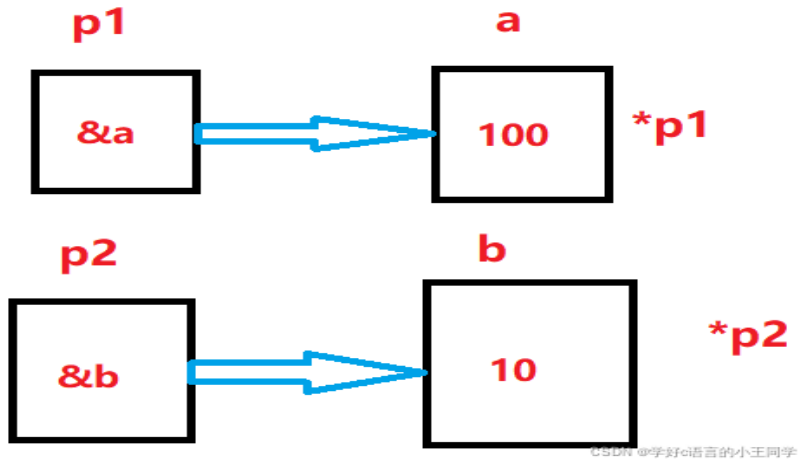
指针的定义与赋值

int y=8;
int *yptr=&y; //正确
char *yptr1=&y; //错误，指针类型与指向变量类型不一致
char yptr2=(char)&y; //正确，通过强制转换，将 int型地址转换为char型，不建议使用




```
int a = 100, b = 10;//定义整型变量a，b并初始化
int* p1, * p2;    //定义指向整型数据的指针变量p1，p2;
p1 = &a;          //把变量a的地址赋给指针变量p1
p2 = &b;          //把变量b的地址赋给指针变量p2
```





本次课的实验介绍

二、实验原理（一图胜千言）

直接引用和间接引用的理解

```
int count=7; //直接引用（访问变量的值）
int *countPtr=&count; //间接引用（访问）
```

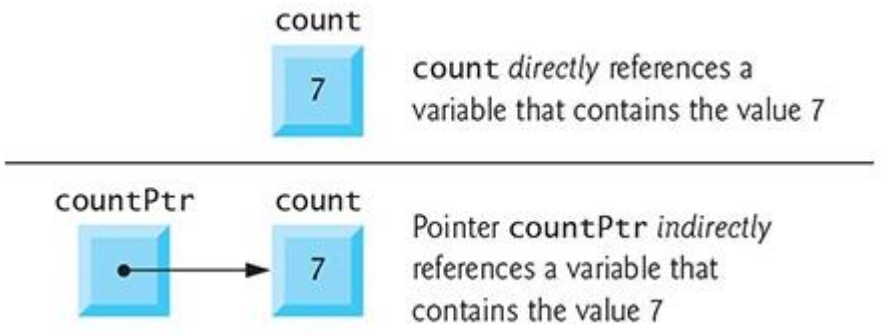


Fig. 8.1 Directly and indirectly referencing a variable

《C++ How to Program》Ninth , p280

Pointer variables contain memory addresses as their values. Normally, a variable directly contains a specific value. A pointer contains the memory address of a variable that, in turn, contains a specific value. In this sense, a variable name directly references a value, and a pointer indirectly references a value (Fig. 8.1).
Referencing a value through a pointer is called indirection. Diagrams typically represent a pointer as an arrow from the variable that contains an address to the variable located at that address in memory.

指针变量包含内存地址作为其值。通常，一个变量直接包含一个特定的值。指针包含一个变量的内存地址，而这个变量又包含一个特定的值。从这个意义上说，变量名直接引用值，指针间接引用值（图8.1）。通过指针引用一个值称为间接引用。图表通常将指针表示为箭头，从包含地址的变量到位于内存中该地址的变量。

本次课的实验介绍

二、实验原理（联系实际现实生活）

由于通过地址能找到所需的变量单元，可以说，地址指向变量单元，打个比方，一个房间的门口挂了一个房间号2008，这个2008就是房间的地址，或者说，2008“指向”该房间。因此，将地址形象化地称为“指针”。意思是通过它能找到以它为地址的内存单元!

现实生活中，通常我们通过房间编号（地址）找人，类似指针（地址）找变量，再找变量的值（变量里的内容）

本次课的实验介绍

二、实验原理

- 取地址运算符 &
- 间接寻址运算符 *

```
// Pointer operators & and *.
#include <iostream>
using namespace std;

int main() {
   int a{7}; // initialize a with 7
   int* aPtr = &a; // initialize aPtr with the address of int variable a

   cout << "The address of a is " << &a
      << "\nThe value of aPtr is " << aPtr;
   cout << "\n\nThe value of a is " << a
      << "\nThe value of *aPtr is " << *aPtr << endl;
}
```

```
The address of a is 002DFD80
The value of aPtr is 002DFD80

The value of a is 7
The value of *aPtr is 7
```

理论课课件上swap实例一定在电脑上调试理解好

二、实验原理

```
#include <iostream>
using namespace std;
int main（）
{ void swap（int *p1,int *p2）;    //函数声明
   int *pointer_1,*pointer_2, a, b;  //定义指针变量pointer_1,pointer_2
   cin>>a>>b;  // 编译阶段a,b 被转换为地址
  pointer_1=&a;                    //使pointer_1指向a
  pointer_2=&b;                    //使pointer_2指向b
if（a<b）
swap（pointer_1,pointer_2）; //如果a<b，使*pointer_1和*pointer_2互换
cout<<"max="<<a<<" min="<<b<<endl;//a已是大数，b是小数
return 0;
}

                                   // 注意！函数形参中*的含义是什么
void swap（int *p1,int *p2）      //函数的作用是将*p1的值与*p2的值交换
{ int temp;
   temp=*p1;                 // 注意！函数体中*的含义
   *p1=*p2;
   *p2=temp;     }
```

二、实验原理

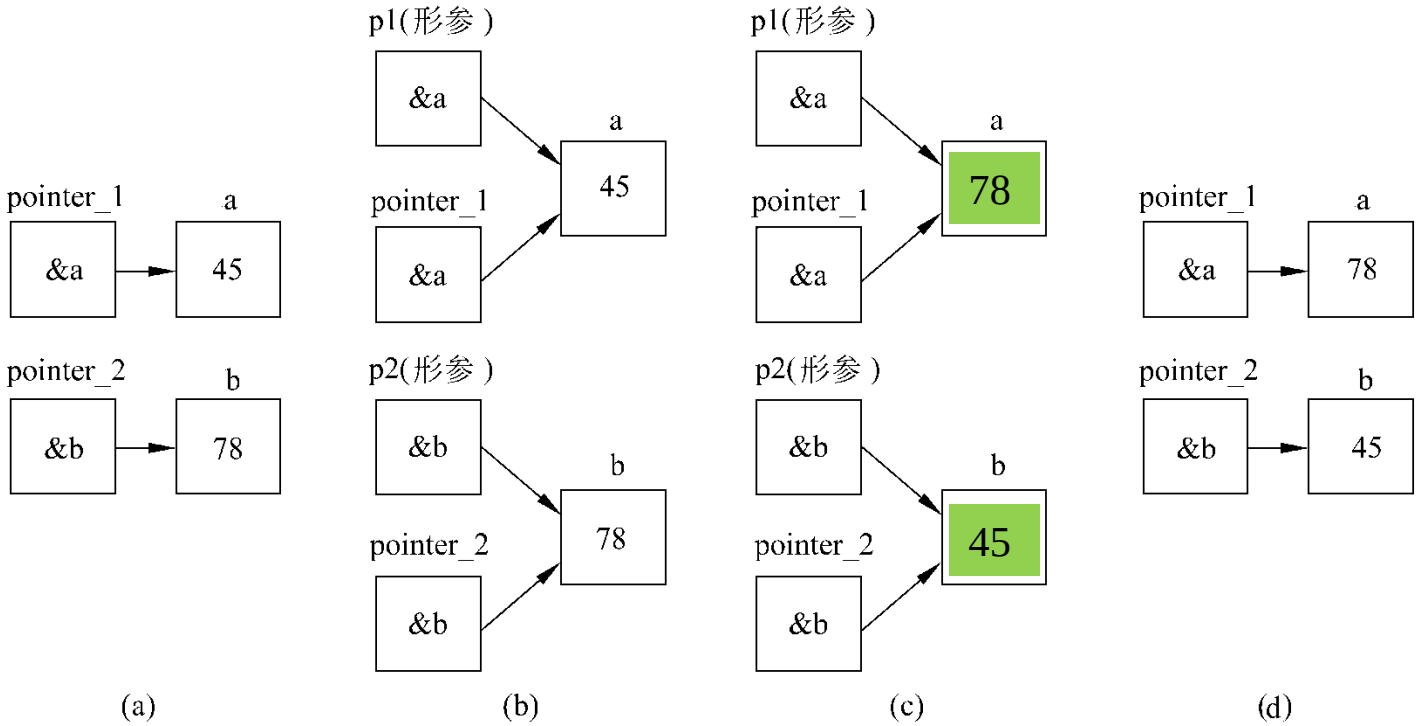
运行情况如下：

45 78↙

max=78 min=45

请注意： 不要将main函数中的swap函数调用写成

if（a<b） swap（*pointer_1,*pointer_2）；



二、实验原理

请注意交换*p1和*p2的值是如何实现的。如果写成以下这样

就有问题了：

```
void swap（int *p1,int *p2）
{           int *temp；
            *temp = *p1；          //此语句有问题，什么问题，调试分析一下
            *p1=*p2;
            *p2=*temp；
}
```

可以看到，在执行swap函数后，主函数中的变量a和b的值改变了。这个改变不是通过将形参值传回实参来实现的。

```
#include <iostream>
using namespace std;
int main ( )
{
        void swap (int *p1,int *p2) ;
        int *pointer_1,*pointer_2,a,b;
        cin>>a>>b;
        pointer_1=&a;
        pointer_2=&b;
        if (a<b) swap (pointer_1,pointer_2) ;
        cout<<"max="<<a<<" min="<<b<<endl;
        return 0;
}

void swap (int *p1,int *p2)
{
        int *temp;   // 错误，改变形参无法影响实参，调试分析
         temp=p1;
        p1=p2;
         p2=temp;
}
```

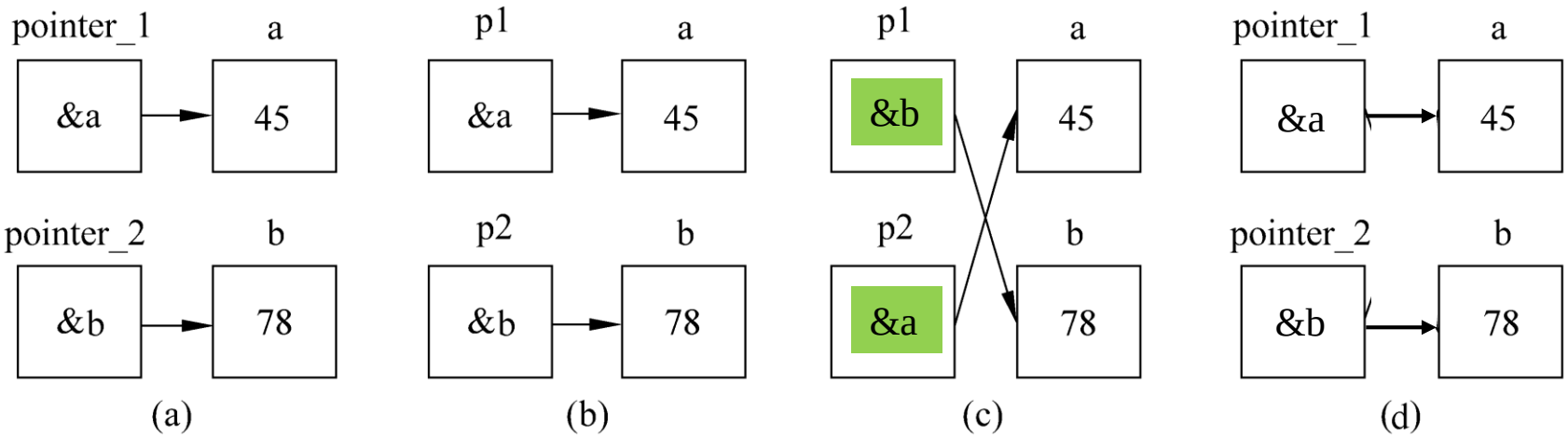


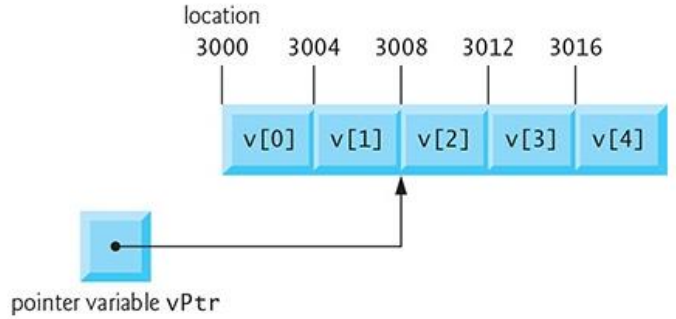
图6.11

实参变量和形参变量之间的数据传递是单向的“值传递”方式。指针变量作函数参数也要遵循这一规则。调用函数时无法改变实参指针变量的值，但可以改变实参指针变量所指向变量的值。

本次课的实验介绍

二、实验原理

指针和数组的关系《C++ How to Program》




```
// Fig. 8.17: fig08_17.cpp
// Using subscripting and pointer notations with built-in arrays.
#include <iostream>
using namespace std;

int main() {
   int b[]{10, 20, 30, 40}; // create 4-element built-in array b
   int* bPtr{b}; // set bPtr to point to built-in array b

   // output built-in array b using array subscript notation
   cout << "Array b displayed with:\n\nArray subscript
notation\n";

   for (size_t i{0}; i < 4; ++i) {
      cout << "b[" << i << "] = " << b[i] << '\n';
   }

   // output built-in array b using array name and
pointer/offset notation
   cout << "\nPointer/offset notation where "
      << "the pointer is the array name\n";

   for (size_t offset1{0}; offset1 < 4; ++offset1) {
      cout << "*(b + " << offset1 << ") =
'\n';
   }

   // output built-in array b using bPtr and array subscr
notation
   cout << "\nPointer subscript notation\n";

   for (size_t j{0}; j < 4; ++j) {
      cout << "bPtr[" << j << "] = " << bPtr[j] << '\n';
   }

   cout << "\nPointer/offset notation\n";

   // output built-in array b using bPtr and pointer/offs
notation
   for (size_t offset2{0}; offset2 < 4; ++offset2) {
      cout << "*(bPtr + " << offset2 << ") = "
         << *(bPtr + offset2) << '\n';
   }
}
```

```
Array b displayed with:

Array subscript notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40

Pointer/offset notation where
 the pointer is the array name
*(b + 0) = 10
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40

Pointer subscript notation
bPtr[0] = 10
bPtr[1] = 20
bPtr[2] = 30
bPtr[3] = 40

Pointer/offset notation
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40
```

本次课的实验介绍

二、实验原理

```
int main()
{
        int arr[10] = { 0 };
        printf("arr= %p \n", arr);
        printf("&arr= %p \n", &arr);
        printf("arr+1= %p \n", arr+1);
        printf("&arr+1= %p \n", &arr+1);
        return 0;
}
```

根据代码我们发现，其实&arr和arr，虽然值是一样的，但是意义应该不一样

实际上&arr表示的是数组的地址而不是数组首元素的地址。

数组的地址+1 跳过整个数组的大小，所以&arr+1相对于&arr的差值是40.

本次课的实验介绍

二、实验原理

VisualStudio中指针变量如何在debug中查看其值

VisualStudio在调试时，指针变量只会显示其地址，其中的数据值我们需要特殊处理才可以看到。

程序如右

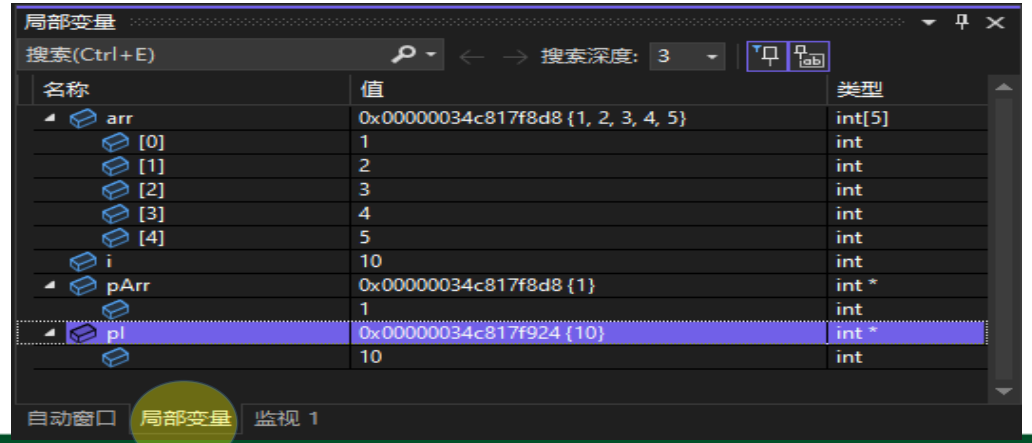
```
#include <stdio.h>

int main()
{
        //定义变量
        int arr[] = { 1,2,3,4,5 };
        int* pArr = arr;
        int i = 10;
        int* pI = &i;

        printf("i=%d\n", i);    //这里设置断点

        return 0;
}
```

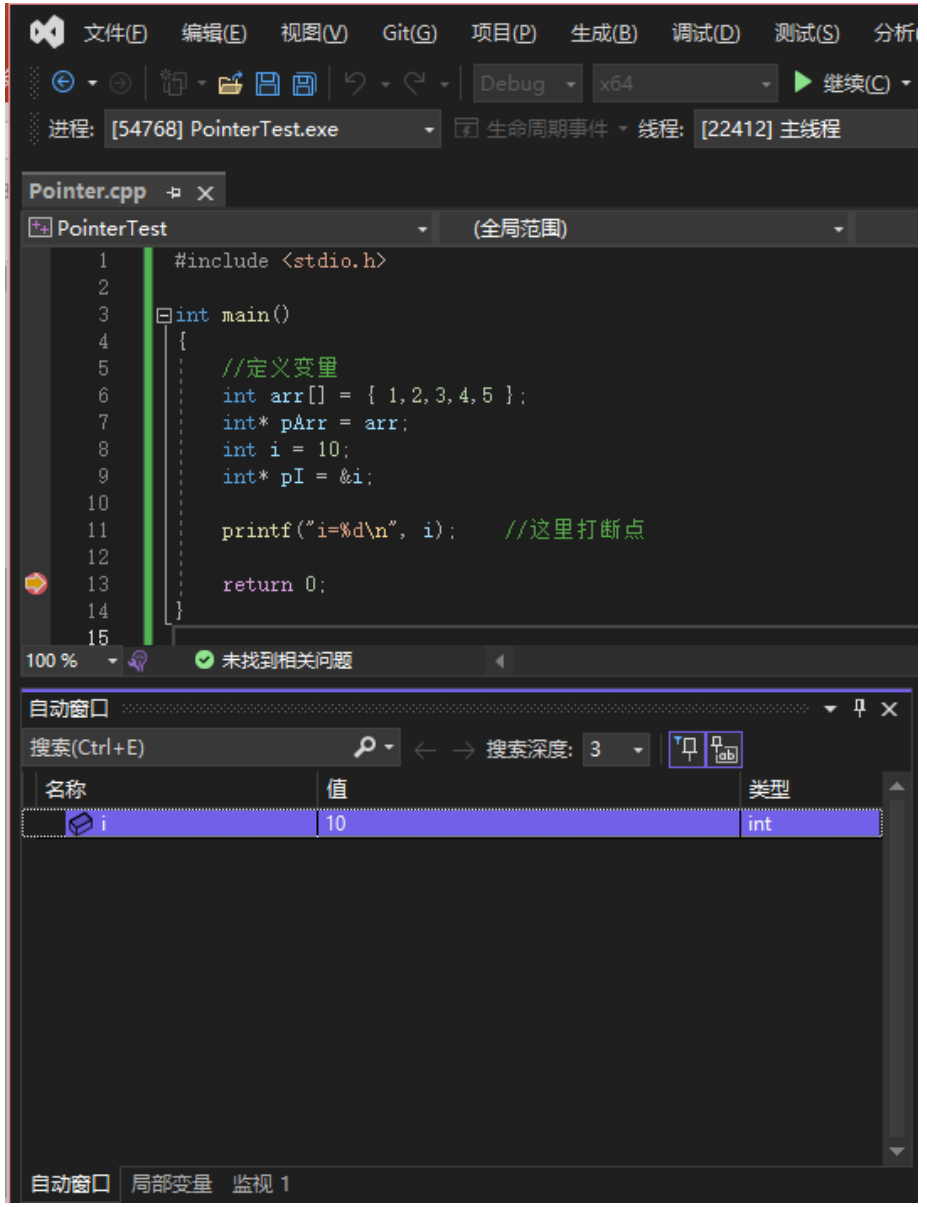
VisualStudio中指针变量如何在debug中查看其值

二、实验原理

- 在printf函数前按F9设置断点，
按F5键执行到断点，只能看到如右图。
- 注意这时左下角对应的是“自动窗口”，只显示变量I
- 这时选择左下角“局部变量”就可以看到指针变量，
我们可以看到指针是64位地址

- 点击变量左侧箭头，可以展开具体变量内容





本次课的实验介绍

二、实验原理

Const常量和常量指针四种情况：

- **a nonconstant pointer to constant data　　非常量指针指向常量数据**
- **a constant pointer to nonconstant data　　常量指针指向非常量数据**
- **a constant pointer to constant data　　常量指针指向常量数据**
- **a nonconstant pointer to nonconstant data　　非常量指针指向非常量数据**

➢ **const int* countPtr;　// a nonconstant pointer to constant data**

The declaration is read from right to left as " countPtr is a pointer to an integer constant" or more precisely, " countPtr is a nonconstant pointer to an integer constant."

声明从右到左读取为"countPtr是指向整数常量的指针"，或者更准确地说，"countPtr是指向整数常量的非常量指针"

本次课的实验介绍

二、实验原理

- **int * const Ptr; //** a constant pointer to nonconstant data
- **const int * const Ptr; //** a constant pointer to constant data
- **int * Ptr; //** a nonconstant pointer to nonconstant data

本次课的实验介绍

二、实验原理 const常量和常量指针

```
// Attempting to modify data through a
// nonconstant pointer to constant data.

void f(int const*); // prototype

int main() {
   int y{0};

   f(&y); // f will attempt an illegal modification
}

// constant variable cannot be modified
through xPtr
void f(int const* xPtr) {
   *xPtr = 100; // error: cannot modify a const
object
}
```

```
// Attempting to modify a constant pointer to nonconstant
data.

int main() {
   int x, y;

   // ptr is a constant pointer to an integer that can be
modified
   // through ptr, but ptr always points to the same memory
location.
   int* const ptr{&x}; // const pointer must be initialized

   *ptr = 7; // allowed: *ptr is not const
   ptr = &y; // error: ptr is const; cannot assign to it a new
address
}
```

本次课的实验介绍

二、实验原理 const常量和常量指针

```
// Attempting to modify a constant pointer to constant data.
#include <iostream>
using namespace std;

int main() {

        int x{5}, y;

  // ptr is a constant pointer to a constant integer.
  // ptr always points to the same location; the integer
  // at that location cannot be modified.
  const int* const ptr{&x};

  cout << *ptr << endl;

  *ptr = 7; // error: *ptr is const; cannot assign new value
  ptr = &y; // error: ptr is const; cannot assign new address
}
```

本次课的实验介绍

二、实验原理（sizeof用于数组和指针）

```
// Sizeof operator when used on a built-in array's name
// returns the number of bytes in the built-in array.
#include <iostream>
using namespace std;

size_t getSize(double*); // prototype

int main() {
   double numbers[20]; // 20 doubles; occupies 160 bytes on our system

   cout << "The number of bytes in the array is " << sizeof(numbers);

   cout << "\nThe number of bytes returned by getSize is "
      << getSize(numbers) << endl;
}

// return size of ptr
size_t getSize(double* ptr) {
   return sizeof(ptr);
}
```

```
The number of bytes in the array is 160
The number of bytes returned by getSize is 4
```

本次课的实验介绍

二、实验原理 函数指针和指针函数

函数指针是指一个指针变量，该指针变量存储了一个函数的地址。通过函数指针可以实现动态调用函数，根据需要在程序运行时指定要调用的函数。函数指针的声明方式为：返回类型 (*指针变量名)(参数列表)。

指针函数则是指一个返回指针的函数，即函数的返回值是一个指针类型。指针函数的声明方式为：返回类型 (*函数名)(参数列表)。

函数指针：int (*operation)(int, int);

指针函数：int* findMax(int* arr, int n);

函数指针实例

```
#include <iostream>
using namespace std;
int add(int a, int b) {return a + b;}
int subtract(int a, int b) {return a - b;}

int main() {
int (*operation)(int, int);//函数指针
int result;
operation = &add;
result = operation(4, 2);
cout << "4 + 2 = " << result << endl;
operation = &subtract;
result = operation(4, 2);
cout << "4 - 2 = " << result << endl;
return 0;
}
```

在这个例子中，定义了两个函数add()和subtract()，它们分别实现加法和减法运算。

接下来定义了一个函数指针变量operation，用于存储函数的地址。通过赋值操作将函数地址分别赋给operation，然后通过函数指针调用函数并计算结果，最后输出结果。

指针函数实例

```
#include <iostream>
using namespace std;
int* findMax(int* arr, int n);//指针函数
int main() {
int arr[] = { 10, 23, 5, 17, 25 };
int* maxPtr = findMax(arr, 5);
cout << maxPtr << endl;
cout << *maxPtr << endl;
if (maxPtr != nullptr) {
cout << "Max value is " << *maxPtr <<
endl;
}else {cout << "Array is empty." << endl;}
return 0;
}
int* findMax(int* arr, int n)
{if (n <= 0) {return nullptr;}
int* max = arr;for (int i = 1; i < n; i++)
{if (arr[i] > *max) {max = &arr[i];}}
return max;
```

在这个例子中，定义了一个指针函数findMax()，用于查找数组中的最大值，并返回指向该最大值的指针。在函数中，先判断数组是否为空，如果为空则返回nullptr。接着定义一个指针变量max，并将其初始化为数组的第一个元素的地址。使用循环遍历数组，如果遇到比max指向的值更大的元素，则将max指向该元素的地址。最后返回max指向的地址，即最大值的地址。

在main()函数中，调用findMax()函数查找数组中的最大值，并将返回的指针赋给maxPtr指针变量。如果返回的指针不为nullptr，则输出最大值。否则输出数组为空的提示信息。

本次课的实验介绍

三、实验内容1

1.使用指针变量对一字符串按照字母，空格、数字和其他字符进行分类统计（提示：读一行字符包括空格用函数cin.getline(ch, 81)）。

1） 输入输出分析及算法；

2） 编程源代码加注释；

3） 调试与测试；

4） 遇到的问题及解决方法；

针对与上述实验题目，同学们先自己独立分析思考，尝试着自己先写算法，代码，调试，然后在网上找其他代码进行优缺点比较。

数据结构、算法设计

- 使用指针变量对一字符串按照字母，空格、数字和其他字符进行分类统计（提示：读一行字符包括空格用函数cin.getline(ch, 81)）
- 数据结构：指针，字符串
- 算法：？？？循环语句、判断选择语句。。。

“多读多写”，“多读”举例：网上找到代码如下：

C语言程序(函数版本)

```
#include <stdio.h>

int main(){
void count(char *);
char str[100];
printf("input string：\n");
gets(str);
count(str);
 return 0;
}

void count(char *string){
char *p;
int i=0,j=0,k=0,w=0,y=0;
 for(p=string;*p!='\0';p++)    {
 if(*p>='A'&&*p<='Z') ++i;
else if(*p>='a'&&*p<='z') ++j;
else if(*p>='0'&&*p<='9') ++k;
else if(*p==' ') ++w;
else          ++y;
}
printf("the number of capital letter:  %d\n",i);
printf("the number of low-case letter:  %d\n",j);
 printf("the number of figure:  %d\n",k);
printf("the number of space:  %d\n",w);
printf("the number of other characters:  %d\n",y);
}
```

代码优缺点:

1）无注释

2）变量命名不规范：标识符的选择应做到“见名知义”、“常用取简”、“专用取繁”

3）

C语言程序(函数版本）

```
#include<stdio.h>
#include <string.h>

void count(char * str)
{
    int i, m=0, n=0, g=0, h=0, r=0;
    for(i=0; *(str+i)!='\0'; i++)
    {
        if (*(str+i) >= 'A' && *(str+i) <= 'Z')
            m = m+1;
        else if (*(str+i) >= 'a' && *(str+i) <= 'z')
            n = n+1;
        else if (*(str+i) == ' ')
            g = g+1;
        else if (*(str+i) >= '0' && *(str+i) <= '9')
            h = h+1;
        else
            r = r+1;
        }
        printf("大写字母：%d\n", m);
        printf("小写字母：%d\n", n);
        printf("空格：%d\n", g);
        printf("数字：%d\n", h);
        printf("其他字符：%d\n", r);
}

int main ()
{
    char str[100];

    printf("请您输入一串字符\n");
    gets(str);
    count( str);

    return 0;
}
```

代码优缺点：

1）无注释

2）

C语言程序(非函数版本)

分析解答:C语言并没有字符串类型，只能采用字符数组或者字符指针的形式来使用字符串。要记住一点，不论我们使用的是字符串常量还是字符串变量，为了方便处理字符串，系统自动 给字符串加上一个结束标志'\0'（'\0'代表ASCII为0的字符，他不是可显示字符，只是一个空操作符，提供标志辨识功能，用它做结束标志不会产生附加的操作或者增加有效字符）。字符串在内存中连续存储，占用一块连续的空间。

```
#include<stdio.h>
#include<stdlib.h>

int main(){

int n, i=0,alpha=0,Alpha=0,digit=0,space=0,other=0;

char *str;

 //输入N，确定字符串长度是多少；然后申请相应长度的地址空间并赋值给str。
scanf("%d",&n);
//  消去scanf函数 遗留下来的换行符 对统计的干扰
getchar();

if((str = (char *)malloc(n * sizeof(char))) == NULL){

printf("Not able to allocate memory");
exit(1);
}

  //输入字符串。这里要注意，字符输入函数的选取，scanf因为不能接受空格，会导致最终统计结果的不准确，故在此选用gets()/getchar();
gets(str);

                //统计字符个数。 字符串以'\0'为终止符在内存中连续存储的，
while(*(str+i) != '\0'){
if(*(str+i) >= 'A' && *(str+i) <= 'Z' ) {  //大小字母
        Alpha ++;
}else if(  *(str+i) >= 'a' && *(str+i) <= 'z'  ){  //小写字母
        alpha ++;
}else if( *(str+i) >= '0' && *(str+i) <= '9' ){   // 数字
        digit ++;
}else if( *(str+i) == ' ' ){   //空格
       space ++;
}else{  //其他字符
        other ++;
}
i++;
}
printf("alpha = %d \n  Alpha = %d\n  digit = %d \n  space=%d\n other=%d \n
",alpha,Alpha,digit,space,other);

                return 0;

}
```

C语言程序

分析解答:C语言并没有字符串类型，只能采用字符数组或者字符指针的形式来使用字符串。要记住一点，不论我们使用的是字符串常量还是字符串变量，为了方便处理字符串，系统自动 给字符串加上一个结束标志'\0'（'\0'代表ASCII为0的字符，他不是可显示字符，只是一个空操作符，提供标志辨识功能，用它做结束标志不会产生附加的操作或者增加有效字符）。字符串在内存中连续存储，占用一块连续的空间。

```
#include<stdio.h>
#include<stdlib.h>
int main(){
int n,i=0,alpha=0,Alpha=0,digit=0,space=0,other=0;
char *str;
 //输入N，确定字符串长度是多少；然后申请相应长度的地址空间并赋值给str。
scanf("%d",&n);
//   消去scanf函数 遗留下来的换行符 对统计的干扰
getchar();

if((str = (char *)malloc(n * sizeof(char))) == NULL){

printf("Not able to allocate memory");
exit(1);
}
 //输入字符串。这里要注意，字符输入函数的选取，scanf因为不能接受空格，
会导致最终统计结果的不准确，故在此选用gets()/getchar();
//也可以使用这个方式输入字符串
                while(i<=n){
                                *(str+i) = getchar();
                                i++;
                }
```

```
//  也可以根据输入的字符个数循环遍历字符串

for(i=0;i<n;i++){

                if(*(str+i) >= 'A' && *(str+i) <= 'Z' ) {  //大小字母
                                Alpha ++;
                }else if(  *(str+i) >= 'a' && *(str+i) <= 'z'  ){  //小写字母
                                alpha ++;
                }else if( *(str+i) >= '0' && *(str+i) <= '9' ){  // 数字
                                 digit ++;
                }else if( *(str+i) == ' ' ){   //空格
                                space ++;
                }else{  //其他字符
                                other ++;
                }
                }

                printf("alpha = %d \n  Alpha = %d\n  digit = %d \n  space=%d\n other=%d
\n ",alpha,Alpha,digit,space,other);

                return 0;

}
```

归纳总结

1，字符串处理在计算机中会经常遇到，因此，了解字符串的定义、初始化、输入输出、存储方式等至关重要。

2，C语言编译系统会自动的在字符串末尾添加'\0'辨识符作为字符串结束的标志，因此，在循环读取字符串中的字符时可以使用这个标志作为循环跳出的判断。

3，为了方便字符串的输入输出，C语言提供了除 scanf()之外的getchar()、gets()函数。可以方便我们对字符串进行输出输出。

4，字符数组的数组名是字符串首地址，是一个常量，不可以对其进行运算，字符串指针是一个指针变量，可以进行运算，因此可以更灵活的操作字符串，提高字符串的处理效率。

C++语言程序（非函数版本）

```
#include<iostream>
#include<string>
#include<cctype>

using namespace std;

int main(void)
{
        char line[81];
        int word,space,digit,other;

        while(cin.getline(line, 81))
        {
                char *p = line;

                word = space = digit = other = 0;

            while(*p!='\0')
            {
                        if(isalpha(p[i]))
                                word++;
                        else if(isspace(p[i]))
                                space++;
                        else if(isdigit(p[i]))
                                digit++;
                        else
                                other++;

                        p++;
                    }
            cout << "Words: " << word << endl;
            cout << "Spaces: " << space << endl;
            cout << "Digits: " << digit << endl;
            cout << "Other charateristics: " << other << endl;
                }
                return 0;
}
```

上述代码是否正确？如不正确，请调试改正。

实验内容2

2.下面的程序中调用了findmax()函数，该函数寻找数组中的最大元素，将该元素的下标通过参数返回并返回其地址值，用指针编程实现findmax()函数。

```
# include <iostream>
using namespace std;
int * findmax(int * array, int size, int * index);
void main ( )
{
int a[10] = {33,91,54,67,82,37,85,63,19,68};
int * maxaddr;
int idx;
maxaddr = findmax(a, sizeof(a)/sizeof( * a), &idx);
cout<<idx<<endl<<maxaddr << endl<<a[idx] << endl;
}
```

1） 输入输出分析及算法；
2） 编程源代码加注释；
3） 调试与测试；
4） 遇到的问题及解决方法

数据结构、算法设计

请思考：？？？

利用指针减少不必要的全局变量

前面贪吃蛇游戏开发中，使用了全局变量。我们可以利用指针减少全局变量

思考如何利用指针去掉全局变量？

```
#include <stdio.h>
Int score=5;//全局变量
void addScore( )
{
        score = score + 1;
}
void minusScore(int *sc)
{
        score = score - 1;
}
void printScore(   )
{
        printf("%d\n", score);
}

int main()
{
        addScore( );
        printScore( );
        minusScore();
        printScore( );
        return 0;
}
```

```
#include <stdio.h>
Int score=5;//全局变量
void addScore( )
{
        score = score + 1;
}
void minusScore(int *sc)
{
        score = score - 1;
}
void printScore(   )
{
        printf("%d\n", score);
}

int main()
{
       addScore( );
        printScore( );
        minusScore();
        printScore( );
        return 0;
}
```

```
#include <stdio.h>
void addScore(int *sc)
{
        *sc = *sc + 1;
}
void minusScore(int *sc)
{
        *sc = *sc - 1;
}
void printScore(int sc)
{
        printf("%d\n",sc);
}

int main()
{
        int score = 5; // 局部变量
        addScore(&score);
        printScore(score);
        minusScore(&score);
        printScore(score);
        return 0;
}
```

利用指针实现动态二维数组

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int high,width,i,j;
        scanf("%d%d",&high,&width);  // 用户自定义输入长
宽

        // 分配动态二维数组的内存空间
        int **canvas=(int**)malloc(high*sizeof(int*));
        for(i=0;i<high;i++)
                canvas[i]=(int*)malloc(width*sizeof(int));

        // canvas可以当成一般二维数组来使用了
        for (i=0;i<high;i++)
                for (j=0;j<width;j++)
                        canvas[i][j] = i+j;
        for (i=0;i<high;i++)
        {
                for (j=0;j<width;j++)
                        printf("%d ",canvas[i][j]);
                printf("\n");
        }

        // 使用完后清除动态数组的内存空间
        for(i=0; i<high; i++)
                free(canvas[i]);
        free(canvas);

        return 0;
}
```

多读：指针数组实现水波纹仿真程序

- https://codebus.cn/contributor/hao-water-ripple-effect
- 按照TOP-DOWN自顶向下设计思路研读
- 学习良好的注释风格
- 学习良好的代码书写格式规范
- 。。。。。。

学习的四重境界

知学、好学、会学、乐学

祝大家实验顺利！