内容回顾

- 指针及其应用-1
  - 指针的基本概念
    - 指针类型的构造
    - 指针变量的定义与初始化
    - 指针数组
    - 多级指针变量
    - 通用指针与void类型
    - 指针类型相关的基本操作

```
int *
float *
double *
char *
```

```
int * pi = &i;
float * pf = &f;
double * px = &x;
char * pch = &ch;
```

```
int * ap[3] = {&i, &j, &k};
```

```
int ** pp = &pi;
```

```
void *
```

```
void * pv = 0;
```

```
* (与&互逆)
=
>  <  >=  <=  ==  !=
+n  -n  ++  --
-
[ ]
```

内容回顾

- 指针及其应用-2
  - 用指针操纵数组
  - 用指针在函数间传递数据
    - 传址调用
    - const
    - 指针型函数**
  - 用指针访问动态变量
    - 动态变量的创建、访问与撤销
    - 内存泄露与悬浮指针**
  - 用指针操纵函数**

```
int * ap[3] = {&i, &j, &k};
//注意ap与q的区别
```

```
int * p = a;
int (*q)[10] = &a;
int (*r)[5][10] = &b;
```

```
Fun(a, …);
void Fun(int *pa)
{ *pa…pa++…}
```

```
Fun(&n);
void Fun(int *pn)
{ *pn = …}
```

```
void Fun(const int *pa)
{ *pa…pa++…}
  *pa = …   ×
```

```
int * Fun(…)
{…return &…}
```

内容回顾

- 指针及其应用-2
  - 用指针操纵数组
  - 用指针在函数间传递数据
    - 传址调用
    - const
    - 指针型函数**
  - 用指针访问动态变量
    - 动态变量的创建、访问与撤销
    - 内存泄露与悬浮指针**
  - 用指针操纵函数**

```
int n; …
int q = new int [n];
delete []q;
q = NULL
```

q自身消亡或指向了别处，但q指向的堆区未撤销；
q自身未消亡、也未指向别处，而q指向的堆区已撤销。

```
int (*pfun1)() = &Fun1;
double (*pfun2)(int) = &Fun2;
```

```
int * Fun(…)
 {…return &…}
//注意Fun与pfun1的区别
```

内容回顾

关于指针的两个操作：“理解两点”

- 当定义了一个指针后：
  - * 和 & “互反”
- 对于数组元素取地址
  - & 和 [0] “互抵消”

  - 例如：

```
double *p, a[10], b[10][10];
p = a; // p=&a[0];
// *p 即是a[0]
p= b[0]; // p = &b[0][0];
double (*q)[10]; // 可存放一个长度为10的数组的地址
q= b; // q = &b[0];
```

C++风格的动态变量！

```
int n, *q=NULL;  //初始化一般放构造函数中
n= …
q = new int[n];  //创建
 ....
q[i] // i=0,...,n-1
 ...

delete []q;          //撤销
q=NULL;          //一般放析构函数中
```

例：对输入的若干个数进行排序，在输入时，先输入各个数，最后输入一个结束标记（如：-1）：

内容回顾

```
const int INCREMENT=10;
int max_len=20,count=0,n,*p=new int[max_len];
cin >> n;
while (n != -1)
{
    if (count >= max_len)
    {       max_len += INCREMENT;
            int *q=new int[max_len];
            for (int i=0; i<count; i++)
                  q[i] = p[i];
            delete []p;
            p = q;
    }
    p[count] = n;
    count++;
    cin >> n;
}
sort(p,count);
……
delete []p;
p =NULL;
```

例：根据要求(输入的值除8的余数)，执行在函数表中定义的某个函数（8个函数中的一个）

内容回顾

```
#include <stdio.h>
#include <cmath>
typedef double (*PF)(double); // 构造函数指针类型
PF func_list[8] = {sin, cos, tan, asin, acos, atan, log, log10}; // 定义长度为8的函数指针的数组
int main( )
{   int index;
    double x;
    do
    {     printf("请输入要计算的函数(0:sin 1:cos 2:tan 3:asin 4:acos 5:atan 6:log 7:log10):\n");
          scanf("%d", &index);
    } while(index < 0 );
    printf("请输入参数：");
    scanf("%lf", &x);
    printf("结果为： %d \n", (*func_list[index%8])(x));
    return 0;
}
```

将一个函数作为另一个函数的形参

内容回顾

例：根据要求计算不同函数的积分

- 还可以把一个函数作为参数传给被调用函数，被调用函数（实参）对应的形参定义为一个函数指针类型，调用时的实参为一个函数的地址。
  - 比如，

```
double Integrate(double (*pfun)(double x), double x1, double x2)
{   double s = 0;
    int i = 1, n;       //i为步长，n为等份的个数，n越大，计算结果精度越高
    printf("please input the precision: ");
    scanf("%d", &n);
    while(i <= n)
    {       s += (*pfun)(x1 + (x2 - x1) / n * i);
            i++;
    }
    s *= (x2 - x1) / n;
    return s;
}
```

例：根据要求计算不同函数的积分

- 函数Integrate可以计算任意一个一元可积函数（由函数指针pfun操纵）在一个区间[x1, x2]上的定积分，该函数的调用形式：

```
double My_func(double x)
{       double f = x;
        return f;
}
```

- Integrate(My_func, 1, 10); //C语言中，一个函数总是占用一段
  // 连续的内存区，而函数名就是该函数所占内存区的首地址
  //计算函数My_func在区间[1, 10]上的定积分
- Integrate(sin, 0, 1);
  //计算函数sin在区间[0, 1]上的定积分
- Integrate(cos, 1, 2);
  //计算函数cos在区间[1, 2]上的定积分

6 复杂数据的描述—构造数据类型
6.3 引用类型

郭延文

2019级计算机科学与技术系

引用类型

- 引用类型用于给一个变量取一个别名。例如：

```
int x=0;
int &y=x; //y为引用类型的变量，可以看成是x的别名
cout << x << ',' << y << endl; //结果为：0,0
y = 1;
cout << x << ',' << y << endl; //结果为：1,1
```

- 在语法上，
  - 对引用类型变量的访问与非引用类型相同
- 在语义上，
  - 对引用类型变量的访问实际访问的是另一个变量（被引用的变量）
  - 效果与通过指针间接访问另一个变量相同

使用注意！

- 对引用类型需要注意下面几点：
  - 定义引用类型变量时，应在变量名加上符号“&”，以区别于普通变量。
    - `int &y=x;`
  - 定义引用变量时必须要有初始化，并且引用变量和被引用变量应具有相同的类型。
    - `int x;`
    - `int &y=x;`
  - 引用类型的变量定义之后，它不能再引用其它变量。
    - `int x1,  x2;`
    - `int &y=x1;`
    - `.....`
    - `y = &x2; //Error`

引用类型作为函数的形参

- 提高参数传递的效率。例如：

```
strcut A
{   int i;
    ……
};
void f(A &x) //x引用相应的实参
{   ……
    … x.i … //访问实参
    ……
}
int main()
{   A a;
    ……
    f(a); //引用传递，提高参数传递效率
    ……
}
```

- 通过形参改变实参的值。例如：

```
#include <iostream>
using namespace std;
void swap(int &x, int &y) //交换两个int型变量的值
{    int t;
     t = x;
     x = y;
     y = t;
}
int main()
{    int a=0,b=1;
     cout << a << ',' << b << endl; //结果为：0,1
     swap(a,b);
     cout << a << ',' << b << endl; //结果为：1,0
     return 0;
}
```

思考

```
#include <iostream>
using namespace std;
void swap(int *x, int *y) //交换两个int *型指针变量的值
{     int *t;
       t = x;
      x = y;
      y = t;
}
int main()
{     int a=0,b=1;
      int *p=&a,*q=&b;
      cout << *p << ',' << *q << endl; //p指向a，q指向b；输出：0,1
      swap(p,q);
      cout << *p <<   ',' << *q << endl; //p指向a，q指向b；输出：0,1
      return 0;
}
```

```
int t;
  t = *x;
*x = *y;
*y = *t;
```

再回顾

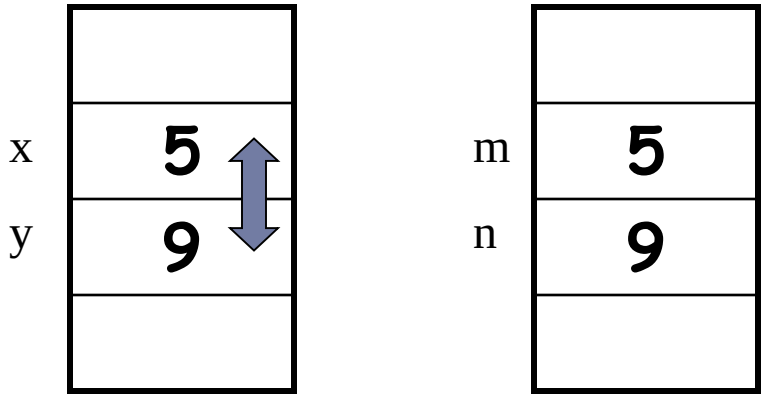
	形参	实参	特点	举例

传值调用	变量 ←	常变量值 表达式值	形参的改变不影响实参	void Swap1(int x, int y)
				{
				int temp = x;
				x = y;
				y = temp;
				}
				int x=m
				int y=n
				int main( )
				{
				int m = 5, n = 9;
				Swap1(m, n);
				printf("%d %d", m, n);
				return 0;
				}

传址调用	指针 ←	地址值	改变形参所指向的变量值来影响实参	void Swap (int *pm, int *pn)
				{
				int temp = *pm;
				*pm = *pn;
				*pn = temp;
				}
				int *pm=&m
				int *pn=&n
				int main( )
				{
				int m = 5, n = 9;
				Swap(&m, &n);
				printf("%d %d", m, n);
				return 0;
				}

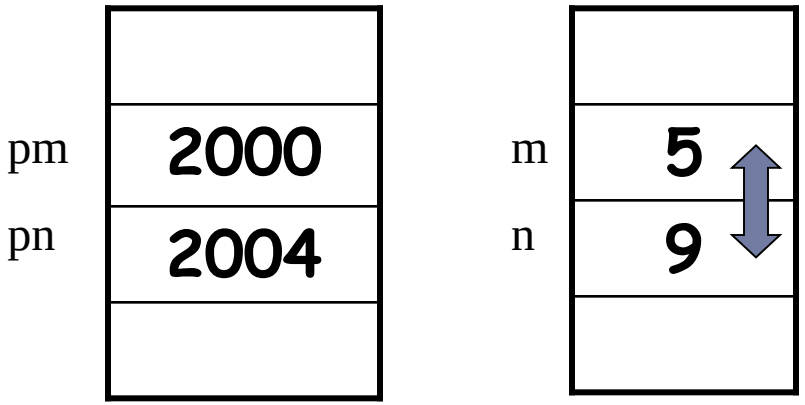
Swap1

再回顾

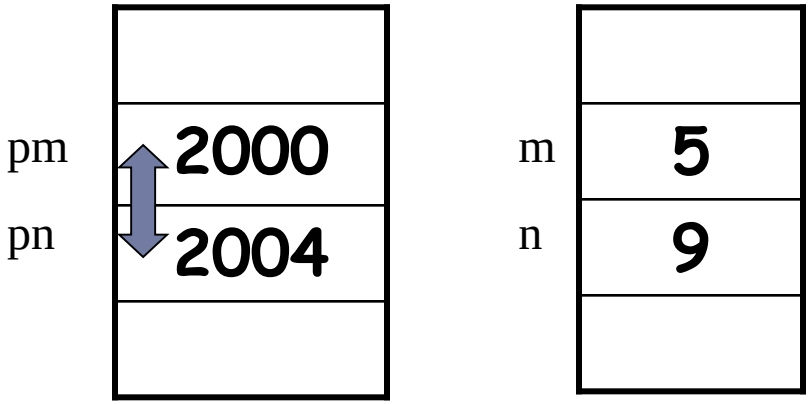



```
void Swap2(int *pm, int *pn)
{
  int *temp = pm;
  pm = pn;
  pn = temp;
}
```

```
int main( )
{
  int m = 5, n = 9;
  Swap2(&m, &n);
  printf("%d %d", m, n);
  return 0;
}
```



Swap



Swap2

要改变实参指针本身的值，
用指针的引用！

```
#include <iostream>
using namespace std;
void swap(int *&x, int *&y) //交换两个int *型指针变量的值
{   int *t;
    t = x;
    x = y;
    y = t;
}
int main()
{  int a=0,b=1;
   int *p=&a,*q=&b;
   cout << *p << ',' << *q << endl; //p指向a，q指向b；输出：0,1
   swap(p,q);
   cout << *p << ',' << *q << endl; //p指向b，q指向a；输出：1,0
   return 0;
}
```

一道相关题目

```
……
void input(Node  *&h) //从表头插入数据，建立链表，h返回头指针
{  int x;
   cin >> x;
   while (x != -1)
   { Node *p=new Node;
     p->content = x;
     p->next =  h;
     h = p;
     cin >> x;
   }
}
int main()
{  Node *head=NULL;
   input( head);
   ……
}
```

```
// Another Option:
Node *input()
{
   Node *h;
   …
   return h;
}
```

常量的引用

- 通过把形参定义成对常量的引用，可以防止在函数中通过引用类型的形参改变实参的值。

```
strcut A
{    int i;
     ……
};
void f(const A &x)
{    x.i = 1; //Error
     ……
}
int main()
{    A a;
     ……
     f(a);
     ……
}
```

```
void f(int *p)
{    ......
     int m;
     p = &m; //OK,
     ... *p ... //通过p可以访问实参以外的数据
}
void g(int &x)
{    int m;
     ......
     x = &m; // Error
     … x … //通过x只能访问实参
}
int main()
{    int a;
     f(&a);
     g(a);
}
```

```
void f(int *const p)
{ ......
   int m;
   p = &m; //Error
   ... *p ... //通过p只能访问实参
}
```

引用类型与指针类型的区别

- 引用类型和指针类型都可以实现通过一个变量访问另一个变量，但在语法上，
  - 引用是采用直接访问形式
  - 指针则需要采用间接访问形式（地址）
- 在作为函数参数类型时，
  - 引用类型参数的实参是一个变量的名字
  - 指针类型参数的实参是一个变量的地址
- 在定义时初始化以后，
  - 引用类型变量不能再引用其它变量
  - 指针类型变量可以指向其它的变量
- 引用类型一般作为指针类型来实现（有时又把引用类型称作隐蔽的指针，hidden pointer）

Q & A