



第七章 指针

函数与指针

指针作为函数参数

指针作为函数返回值

指向函数的指针

函数之间数据交换的基本方式

结构化编程中，C程序由函数构成，程序由`main`启动，函数之间互相调用，完成整个程序的功能。一个函数`fa`调用另一个函数`fb`，相当于请求`fb`来完成某个功能，一般来说`fa`需要传递一些信息给`fb`，然后接收`fb`完成的结果。



函数 fa 传递信息给函数 fb 的方式:

- (1) 通过函数参数 (传值: 直接传递需要信息)
- (2) 通过公共区域交换, fa 将信息写在某个区域, 让 fb 去使用.
 - ✓ 全局变量, fb 直接使用变量名
 - ✓ fb 无法直接访问时, 传递区域地址 (传值, 只不过是地址)
- (3) 引用



函数 fb 返回信息给函数 fa 的方式:

- (1) 通过 $return$ 直接返回信息 (只能返回1个)
- (2) 通过公共区域交换, fb 将信息写在某个区域, 让 fa 去使用。

- ✓ 全局变量, fa 直接使用变量名

- ✓ fa 传递公共区域地址给 fb

- ✓ fb 返回区域地址

- (3) 引用



1. 指针作为函数的参数



#include <stdio.h>

例 实现通过函数交换两个变量的值.

```
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, 整型变量  
a,b  
cin>>a>>b;  
pointer_1=&a; //使pointer_1指向a  
pointer_2=&b; //使pointer_2指向b  
swap(pointer_1,pointer_2); //如果a<b, 使*pointer_1和*pointer_2互换  
printf("a=%d,b=%d", a, b); //a已是大数, b是小数  
return 0;  
}
```

```
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;  
*p1=*p2;  
*p2=temp;  
}
```

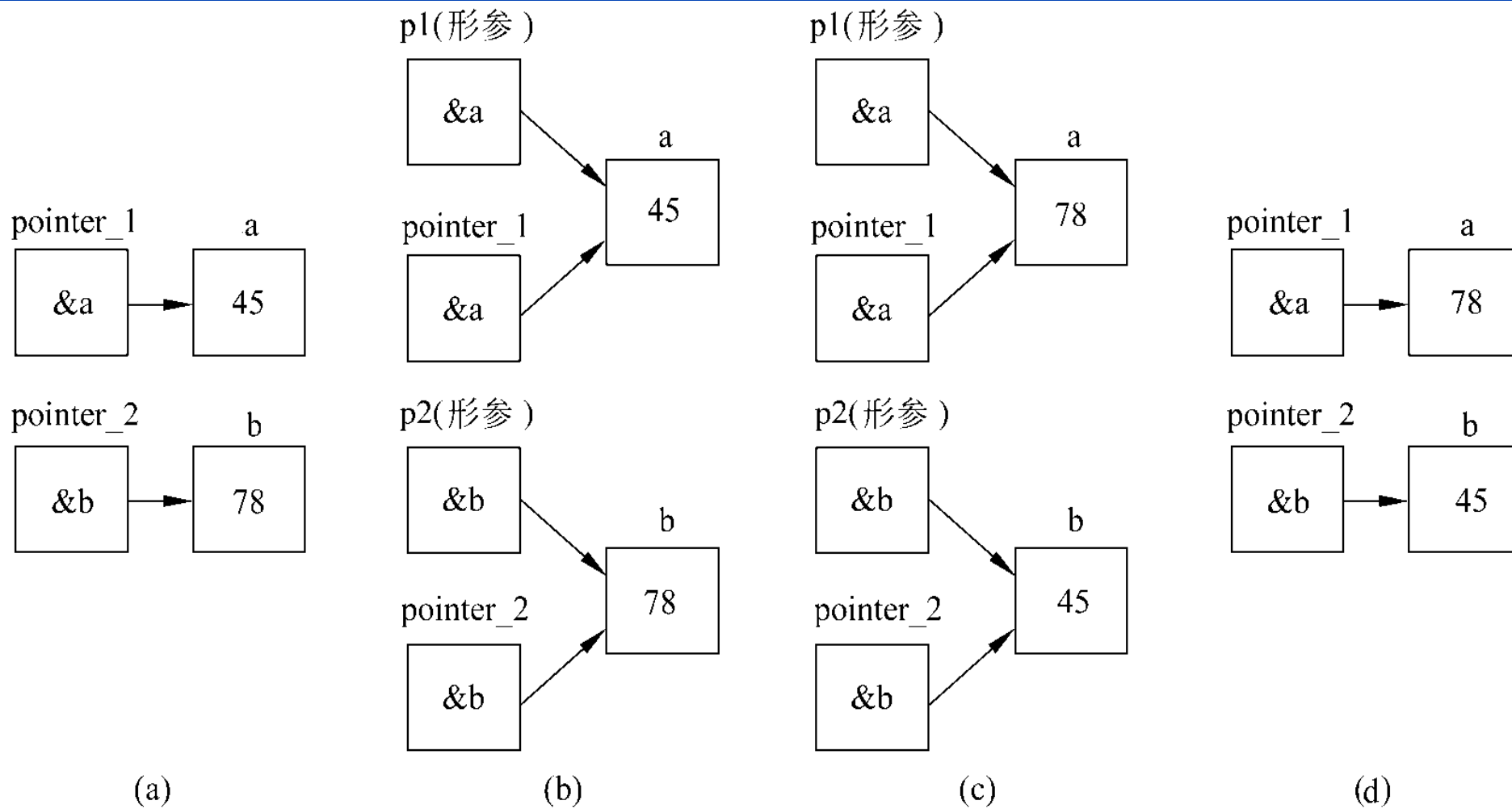


河海大学

运行情况如下:

45 78✓

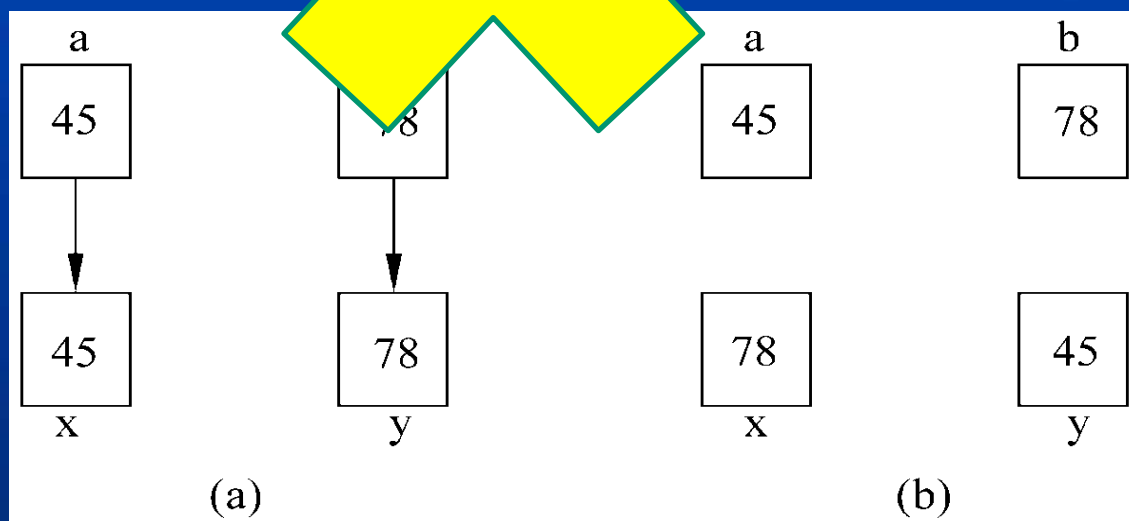
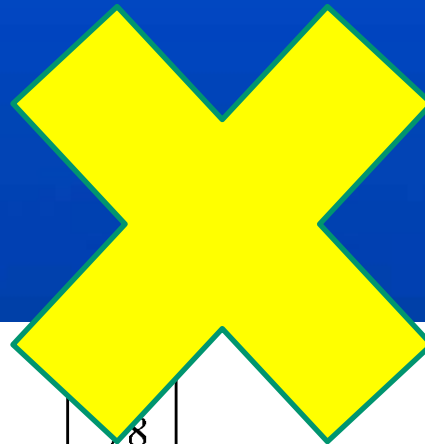
a=78 b=45





请考虑一下能否通过调用下面的函数实现a和b互换。

```
void swap (int x, int y)
{int temp;
temp=x;
x=y;
y=temp;
}
```





形参指针变量的定义

(1) 各种基本类型变量的指针

```
T* p1; T a[10]; //int* p1; int a[10];
```

(2) 指向数组的指针，重要的是数组元素个数

```
T (*p)[n]; T a[m][n]; //int (*p)[4]; int a[2][4];
```

(3) 指针数组

```
T* p[n]; //int* p[5];
```

(4) 多级指针

```
T*** p; //int*** p;
```

```
int i = 100, *p1, **p2, ***p3;  
p1 = &i;  
p2 = &p1;  
p3 = &p2;  
cout << ***p3; //100
```



形参指针变量的定义

(1) 以一维数组形式定义变量指针

void f(T a[]); //a为T* 类型

void f(T a[10]); //a为T* 类型，数组的元素个数不起作用

void f(T* a);

在形参中定义的a，其类型为T*，[]中可以指定数组元素个数，也可以不指定。在调用f时，系统只是分配一个T*的变量空间，而不会实际分配一个数组。

在一维数组中，不论数组元素个数的多少，T a[n]; a都是T*类型



例 写函数求一维数组的均值。

例 用选择法对一维数组元素从小到大排序。

例 写函数求一维数组的均值。

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

```
double mean(int *p, int num){
```

```
    double sum = 0;
```

```
    for(int i=0; i<num; i++){
```

```
        sum += p[i]; //sum += *(p+i);
```

```
    }
```

```
    return sum/num;
```

```
}
```

```
int main(){
```

```
    int a[] = {1,2,3,4,5};
```

```
    double m = mean(a, 5);
```

```
    printf("mean=%d", m);
```

```
}
```

例 写函数求一维数组的均值。

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

```
double mean(int p[], int num) //double mean(int p[5], int num)
{
```

```
    double sum = 0;
```

```
    for(int i=0; i<num; i++){
```

```
        sum += p[i]; //sum += *(p+i);
```

```
    }
```

```
    return sum/num;
```

```
}
```

```
int main(){
```

```
    int a[] = {1,2,3,4,5};
```

```
    double m = mean(a, 5);
```

```
    printf("mean=%d", m);
```

```
}
```



(2) 以二维数组形式定义形参变量指针

void f(T a[][n]); //一定要指定n,因为要确定a+1的操作

void f(T (*p)[n]); //一定要指定n,因为要确定p+1的操作

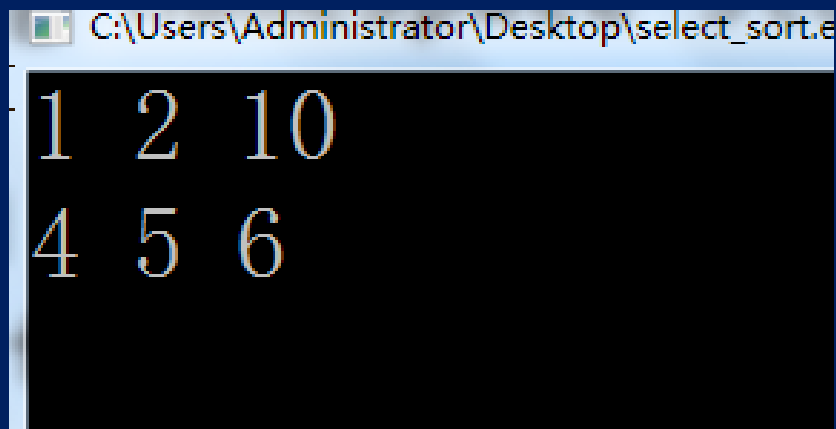
在形参中定义的a，其类型为T(*)[n]，n一定要指定，因为确定后面a+1实际增加的字节数。

例 编写函数，输出二维函数的元素。

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

```
void print(double a2[][3]){  
    for(int i=0; i<2; i++){  
        for(int j=0; j<3; j++){  
            printf("%d", a2[i][j]);  
        }  
        printf("\n");  
    }  
}
```

```
int main(){  
    double a[][3] = {1,2,10,4,5,6};  
    print(a);  
}
```



```
C:\Users\Administrator\Desktop\select_sort.e  
1 2 10  
4 5 6
```


指向数组的指针。数组元素（个数和类型）一定要相同

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

```
void print(double a2[][4]){  
    for(int i=0; i<2; i++){  
        for(int j=0; j<3; j++){  
            printf("%d ", a2[i][j]);  
        }  
        printf("\n");  
    }  
}
```

```
void main(){  
    double a[][3] = {1,2,10,4,5,6};  
    print(a);  
}
```

```
1 #include <iostream>  
2 using namespace std;  
3  
4 void print(double a2[][4]){  
5     for(int i=0; i<2; i++){  
6         for(int j=0; j<3; j++){  
7             cout << a2[i][j] << " ";  
8         }  
9         cout << endl;  
10    }  
11 }  
12  
13  
14 int main(){  
15     double a[][3] = {1,2,10,4,5,6};  
16     print(a);  
17 }
```

[Error] cannot convert 'double (*)[3]' to 'double (*)[4]' for argument '1' to 'void print(double (*)[4])'

例 编写函数返回二维数组所有元素的和。

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

```
double sum(double* p, int c){  
    double s = 0;  
    for(int i=0; i<c; i++){  
        s += p[i];  
    }  
    return s;  
}
```

```
int main(){  
    double a[][3] = {1,2,10,4,5,6};  
    cout << sum(a[0], 6);  
}
```

二维数组的所有元素是按照顺序在内存中存储的，因此，只要传递第一个元素的地址就可以访问到所有元素。但是，此时行列的逻辑位置不容易计算，或者说没有行列的概念。

例 给定二维数组首地址，返回指定行列的值。

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

```
//给定二维数组首地址，返回指定行列的值。
```

```
double getElement(double* p, int r, int c){
```

```
    double *p1 = 0;
```

```
    p1 = p + r*3; //让p1指向第r行
```

```
    p1 += c; //让p1指向第c列
```

```
    return *p1;
```

```
}
```

```
int main(){
```

```
    double a[][3] = {1,2,10,4,5,6};
```

```
    printf("%d", getElement(a[0], 1,0) );
```

```
}
```

获得二维数组的首地址，以及二维数组每行的元素个数，可以计算出每个元素的地址。



2. 指针作为函数的返回值



指针作为函数的返回值

例 编写函数，返回数组最大值元素的地址。

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

```
float* getMax(float* p, int num){  
    float max = p[0];  
    float* pMax = p;  
  
    for(int i=0; i<num; i++){  
        if (max<p[i]){  
            max = p[i];  
            pMax = p+i;  
        }  
    }  
    return pMax;  
}  
  
int main(){  
    float f[] = {1,2,319,12};  
    float* max = getMax(f, 4);  
    printf("%d", *max);  
}
```

```
int main(){  
    float f[] = {1,2,319,12};  
    printf("%d", *getMax(f, 4);  
}
```

函数返回的指针作为临时量与*结合。

返回f1, f2, f3数中最大数的地址

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

```
//返回p1,p2,p3指向的数中最大数的地址
```

```
float* getMax(float* p1, float* p2, float* p3){  
    float* pMax = p1;  
    if (*pMax < *p2){  
        pMax = p2;  
    }  
    if (*pMax < *p3){  
        pMax = p3;  
    }  
    return pMax;  
}  
  
int main(){  
    float f1 = 200.3, f2 = 405.7, f3 = 930;  
    printf("%d", *getMax(&f1, &f2, &f3));  
}
```


返回p1, p2, p3数中最大数的地址

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

```
//返回f1,f2,f3指向的数中最大数的地址
```

```
float* getMax(float f1, float f2, float f3){
```

```
    float max = f1;
```

```
    if (max < f2)
```

```
        max = f2;
```

```
    if (max < f3)
```

```
        max = f3;
```

```
    return &max;
```

```
}
```

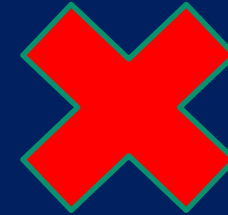
```
int main(){
```

```
    float f1 = 200.3, f2 = 405.7, f3 = 930;
```

```
    float* pf = getMax(f1, f2, f3);
```

```
    printf("%d", *pf);
```

```
}
```



函数内的局部变量在函数退出后被释放了，因此pf指向的是已经被释放的内存，因此不能通过指针访问该内存了。

[Warning] address of local variable 'max' returned [-Wreturn-local-addr]



被调用函数返回的指针指向的内存空间，在被调用函数结束后要仍然是归程序所有，不能被释放了；

如果被释放了，则在被调用函数结束后，虽然可以通过指针访问该区域，但是该区域已经不属于程序了，因此会出现异常结果。



3. 指向函数的指针

函数的代码在内存中占用一块存储区域，该区域的编号最小的地址为该函数的地址，函数的地址称为函数指针，可以定义指针变量存储函数指针。

- ✓ 先写函数原型；
- ✓ 将函数名替换成(*函数指针变量)。



指向函数的指针

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

```
int mysum(int a, int b){  
    return a+b;  
}
```

```
int main(){
```

```
    int (*pf)(int, int);
```

```
    pf = mysum; //pf= &mysum; 也可以  
    printf("%d", (*pf)(3,4)); //pf(3,4);也可以  
    return 0;
```

```
}
```

指向函数的指针



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

```
void sum(int a, int b){
```

```
    //...
```

```
}
```

```
void sum2(float a, float b){
```

```
    //...
```

```
}
```

```
int main(){
```

```
    void (*pf)(int, int); //定义函数指针变量
```

```
    pf = sum; //函数名就代表了函数指针
```

```
    pf = sum2;
```

```
[Error] invalid conversion from 'void (*)(float, float)' to 'void (*)(int, int)' [-fpermissive]
```

```
}
```



```
int main(){
    int a[] = {1,2,0,5,4};
    select_sort(a, 5, isbigger);
    for(int i=0; i<5; i++)
        cout << a[i] << endl;
    select_sort(a, 5, isbigger2);
    for(int i=0; i<5; i++)
        cout << a[i] << endl;

    return 0;
}
```

接上一页：
利用选择排序法进行排序，注意程序中，选择排序法的总流程是不变的；可以通过定义不同的比较两个元素大小的函数来控制排序法的最终结果。

```
bool isbigger(int a, int b){  
    return a>b;  
}  
bool isbigger2(int a, int b){  
    return a<b;  
}
```

```
typedef bool (*PF)(int, int);
```

//typedef定义类型别名。先定义变量，然后加上typedef，变量名就变成类型别名！

```
void select_sort(int* p, int n, PF isbg){  
    for(int i=0; i<n-1; i++){  
        for(int r=i+1; r<n; r++){  
            if (isbg(p[i], p[r])){  
                int t = p[i];  
                p[i] = p[r];  
                p[r] = t;  
            }  
        }  
    }  
}
```

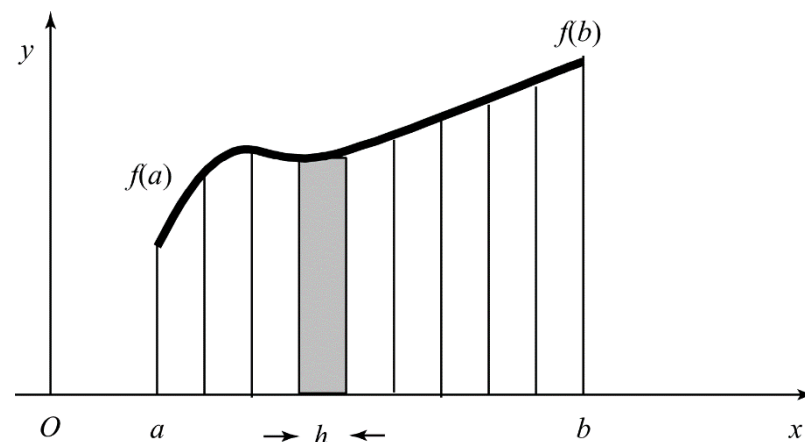
利用选择排序法进行排序，注意程序中，选择排序法的总流程是不变的，可以通过定义不同的比较两个元素大小的函数来控制排序法的最终结果。

函数指针

- 求下列函数的定积分

$$y_1 = \int_0^1 (1 + x^2) dx$$

$$y_2 = \int_0^3 \frac{x}{1 + x^2} dx$$



$$\begin{aligned} \int_a^b f(x) dx &= \frac{h}{2} [f(a) + f(a+h)] + \frac{h}{2} [(f(a+h) + f(a+2h))] + \cdots + \frac{h}{2} [f(a+(n-1)h) + f(b)] \\ &= \frac{h}{2} [f(a) + 2f(a+h) + 2f(a+2h) + \cdots + 2f(a+(n-1)h) + f(b)] \\ &= h \left[\frac{1}{2}(f(a) + f(b)) + \sum_{i=1}^{n-1} f(a+ih) \right] \end{aligned}$$

函数指针

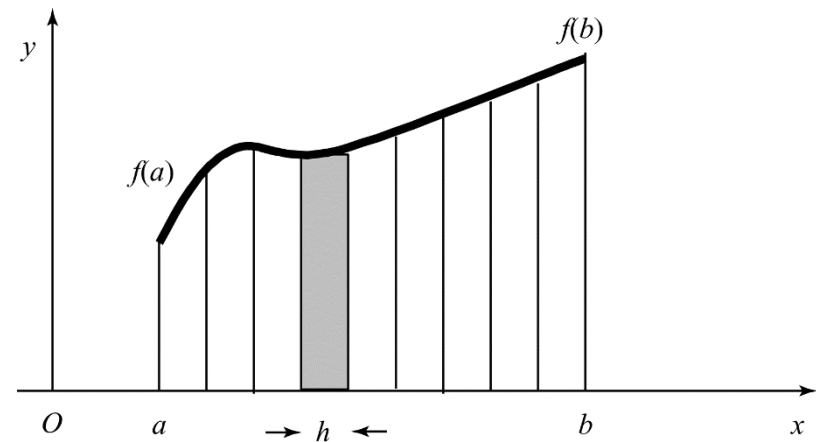
- 不用函数指针的编程方法

```
float IntegralFun1(float a, float b)
{
    float s,h,y;
    int    n,i;

    s = ((1.0+a*a) + (1.0+b*b)) / 2.0;
    n = 100;
    h = (b - a) / n;

    for (i=0; i<n; i++)
    {
        y = a + i * h;
        s += 1.0 + y * y;
    }
    return s * h;
}
```

$$y_1 = \int_0^1 (1+x^2)dx$$



函数指针

- 不用函数指针的编程方法

```
float IntegralFun2(float a, float b)
```

```
{
```

```
    float s,h,y;
```

```
    int    n,i;
```

```
    s = (a/(1.0+a*a) + b/(1.0+b*b)) / 2.0;
```

```
    n = 100;
```

```
    h = (b - a) / n;
```

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

```
    {
```

```
        y = a + i * h;
```

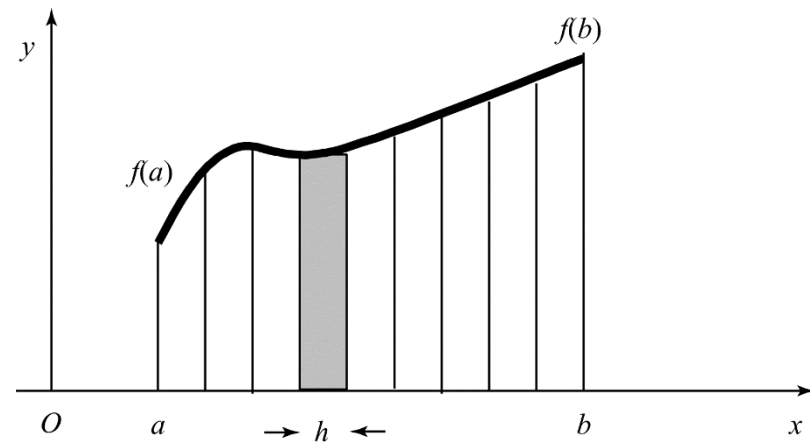
```
        s += y / (1.0 + y * y);
```

```
    }
```

```
    return s * h;
```

```
}
```

$$y_2 = \int_0^3 \frac{x}{1+x^2} dx$$



函数指针

- 使用函数指针的编程方法

```
y1 = Integral(Fun2, 0.0, 3.0);  
y2 = Integral(Fun1, 0.0, 1.0);
```

```
float Integral(float (*f)(float), float a, float b)  
{  
    float s, h, y;  
    int n, i;  
  
    s = ((*f)(a) + (*f)(b)) / 2.0;  
    n = 100;  
    h = (b - a) / n;  
    for (i=0; i<n; i++)  
    {  
        y = a + i * h;  
        s += (*f)(y);  
    }  
    return s * h;  
}
```

```
float Fun1(float x)  
{  
    return 1+x*x;  
}
```

```
float Fun2(float x)  
{  
    return x/(1+x*x);  
}
```

4.命令行参数

- GUI界面之前，计算机的操作界面都是字符式的命令行界面（DOS、UNIX、Linux）
- 通过命令行参数，使用户可以根据需要来决定程序干什么、怎么干
- `main(int argc, char* argv[])`
 - 当你把main函数写成这样时
 - `argc`的值为：参数的数目+1
 - `argv[0]`为指向命令名的字符指针
 - `argv[x] (x>1)`为指向每个参数的字符指针

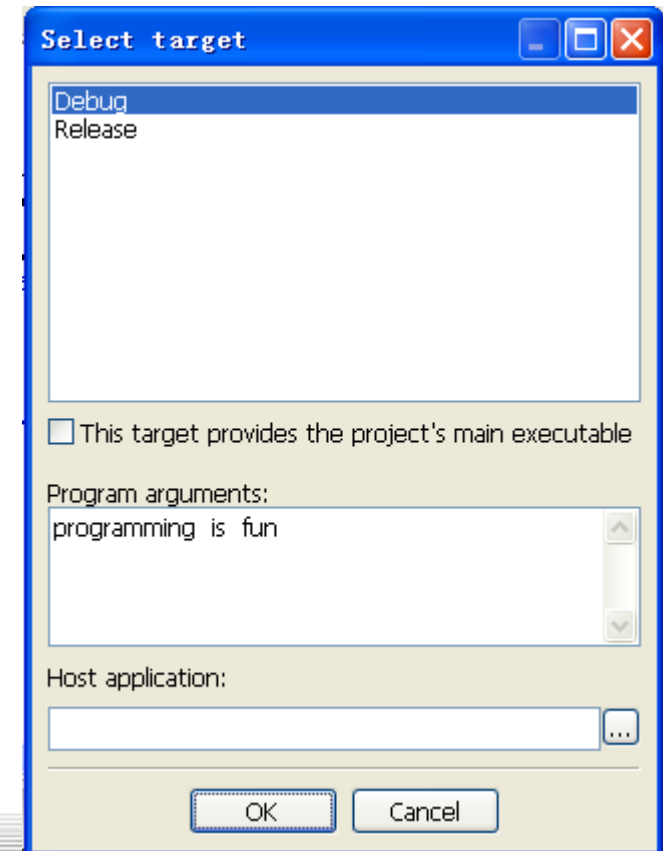
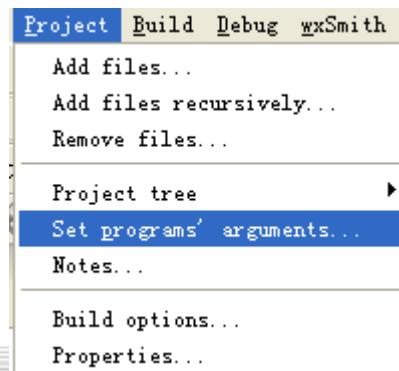
演示命令行参数与main函数各形参之间的关系

```
int main(int argc, char *argv[])
{
    int i;
    printf("The program name is:%s\n", argv[0]);
    if (argc > 1)
    {
        printf("The other arguments are\n");
        for (i = 1; i<argc; i++)
        {
            printf("%s\n", argv[i]);
        }
    }

    return 0;
}
```

输入命令行

hello.exe programming is fun



演示命令行参数与main函数各形参之间的关系

```
main(int argc, char *argv[])
{
    int i;
    printf("The program name is:%s\n", argv[0]);
    if (argc > 1)
    {
        printf("The other arguments are following:\n");
        for (i = 1; i<argc; i++)
        {
            printf("%s\n", argv[i]);
        }
    }
}
```

输入命令行

hello.exe programming is fun




输出

The program name is: hello.exe

**The other arguments are following:
programming
is
fun**

作 业

1. 编写程序实现一个简单的四则运算的程序。
 2. 编写函数，实现对一个给定首地址的整型数组元素进行排序。要求排序顺序能够通过比较函数决定。
 3. 编写函数，实现对给定首地址的两个数组元素进行从小到大合并，给定数组元素已经按照从小到大排序。
- 



河海大學

欢迎交流

