指针【重点】

指针的重要性

表示一些复杂的数据结构

快速的传递数据，减少内存的耗用【重点】

使函数返回一个以上的值【重点】

能直接访问硬件

能够方便的处理字符串

是理解面向对象语言中引用的基础

总结:指针是 C 语言的灵魂。

指针的定义

地址

内存单元的编号

从零开始的非负整数

范围:【0~4G-1】

指针

指针就是地址，地址就是指针

指针变量就是存放内存单元编号的变量，或者说指针变量就是存放地址的变量

指针和指针变量是两个不同的概念

但是要注意通常我们叙述是会把指针变量简称为指针，实际他们含义并不同

指针的本质就是一个操作受限的非负整数

指针的分类

①、基本类型指针[重点]

[指针热身程序_2]

```c
#include <stdio.h>
int main(void){
    int * p;    //p是变量的名字，int * 表示p变量存放的是int 类型变量的地址。
                //不表示定义了一个叫做*p的整型变量
                //int * p; 是一个声明int *是数据类型；p是变量的名字
                //所谓(int *);类型就是存放int型变量的地址的数据类型
    int i = 3;
    int j;

    p = &i;
    /*
        1、p保存了i的地址，因此p指向i
        2、p不是i，i也不是p，更准确的说，修改p的值不影响i的值，修改i的值也不会影响p的值。
        3、如果一个指针变量指向某个普通变量，则
            *指针变量就 等价于 普通变量
            例子：
                如果p是个指针变量，并且p存放普通变量i的地址
                则p指向了普通变量i
                *p 就完全等同于 i（*p就是指针变量p存放的那个普通变量地址内的普通变量）
                    或者说：在所有出现*p的地方都可以替换成i
                            在所有出现i的地方都可以替换成*p
                *p 就是以p的内容为地址的变量
    */
    j = * p;
    printf("i = %d\n", i);
    printf("j = %d\n", j);
    return 0;
}
```

[经典指针程序_互换两个数字]

```c
/*
    2020年12月25日
    要想改变主函数的值：一定接受主函数的变量地址
    看懂huhuan_1,huhuan_2为什么不能实现数值互换。
    明白这个程序各个函数改掉那个位置，输出的值有什么变化，知道为什么不能互换。
*/
#include <stdio.h>
/*
    函数声明的时候基本不写形参
    因为形参不重要
*/
void huhuan_1(int, int);
void huhuan_2(int *, int*);
void huhuan_3(int *, int *);
int main (void){
    int a = 3;
    int b = 5;
    /*
    int t;
    t = a;
    a = b;
    b = t;
    */
    huhuan_1(a, b);
    printf("a = %d,  b = %d\n", a, b);
    huhuan_2(&a, &b);//huhuan_2(*p, *q);是错误的 huhuan_2(a, b);也是错误的
    printf("a = %d,  b = %d\n", a, b);
    huhuan_3(&a, &b);
    printf("a = %d,  b = %d\n", a, b);
    return 0;
}
//不能完成互换功能
void huhuan_1(int a, int b){
    int t;
    t = a;
    a = b;
    b = t;//只改变了形参的值，没改变实参的值。
}
//不能完成互换功能
void huhuan_2(int * p , int * q){
    int * t;//如果要互换pq的值，则t一定是int *；不能是int，否则会出错。
    t = p;
    p = q;
    q = t;/*
            huhuan_2函数只互换的pq内存放的地址，并不是互换了a，b的值。
            只改变了pq的值，没有改变*p和*q的值
            只是改变了形参的值，不会影响实参的值
          */
}
//可以完成互换功能
void huhuan_3(int * p, int * q){
    int t;//若果要互换*p和*q的值，则t必须定义成int，不能定义成int *；否则语法出错
    t = *p;//p是int *；*p是int *p代表的是以p的内容为地址的变量
    *p = *q;
    *q = t;
}

```

附注：

*的含义：

1、乘法

2、定义指针

int * p;

//定义了一个名字叫 p 的变量，int *表示 p 只能存放 int 变量的地址

3、指针运算符

该运算符放在已经定义好的指针变量的前面；

如果 p 是一个已经定义好的指针变量

则*p 表示：以 p 的内容为地址的变量。

如何通过被调函数修改主调函数普通变量的值

1）、实参必须为该普通变量地址；

2）、形参必须为指针变量；

3）、在被调函数中通过

*形参名 = ...

的方式可以修改被调函数相关变量的值。

②、指针和数组

指针和一维数组

一维数组名

一维数组名是个指针常量

它存放的是一维数组第一个元素的地址。

[数组_1.cpp]

```
#include <stdio.h>
int main(void){
    int a[5];//a是数组名，5是数组元素的个数，元素就是变量：a[0]~a[4].
//  int a[3][4];//3行4列：a[0][0]是第一个元素，a[i][j]是第i+1行j+1列的元素
    int b[5];
//  a = b;//error a是常量
    printf("%#X\n", &a[0]);  //%#X十六进制形式输出
    printf("%#X\n", a);
    return 0;
}
/*
    在dev c++中的输出结果是：
    ------------------------
    0X62FE00
    0X62FE00
    ------------------------
    总结：
        一维数组名
            一维数组名是个指针常量
            它存放的是一维数组第一个元素的地址。
*/
```

下标和指针的关系

如果 p 是个指针变量，则

p[i] 永远等价于 *(p+i)

```
/*
a[6] = {1,2,3,4,5,6};
a[0]  a[1]  a[i]    i就是下标
*/
```

[指针和数组下标. cpp]

```
#include <stdio.h>
f(int * p, int len){//p接受a数组的第一个元素的地址，p和a也是同一数组;所以p[i] = a[i].
    p[3] = 88;//因为指针变量p存放的是数组a的首地址，所以p[3]用星号表达式表示为*(p+3);即在p[0]的基础上偏移
    printf("%d\n", *(p+3));
}
int main(void){
    int a[5] = {1, 2, 3, 4, 5};
    f(a, 5);//实参a是数组名也是数组内第一个元素的地址 5表示数组长度
    printf("%d\n", a[3]);/*
                        p[i] = a[i] = *(p+i);p和a是同一数组，
                        p[3]、a[3]表示的是第4个元素，
                        *(p+3)表示的是p[0]向右偏移3 (+3)表示偏移量
                     */
    return 0;
}
/*
    总结:
    只要指针变量里存放着数组名，
    也就存放着数组的首地址，
    此时指针变量名就是数组名。
    此时数组与指针相等;
    所以p[i]=a[i];
    p[i]的值用指针变量表示就是*(p+i)---就是指针变量p存放的地址向偏移i。因为指针变量刚开始存放的地址是数组的首地址a[0];
*/
```

确定一个一维数组需要几个参数【如果一个函数要处理一个一维数组，则需要接受该数组的哪些信息】

需要两个参数:(参数的类型)

数组第一个元素的地址；(指针类型的变量)

数组的长度。(整型)

[确定一个数组需要几个参数_3.cpp]

```
#include <stdio.h>
//f函数可以输出任何一个一维数组的内容
void f(int * p, int a){
    int i;
    for (i=0; i<a; i++){
        printf("%d   ", *(p+i));// *(p+i) 等价于 p[i] 等价于 a[i] 也等价于 *(a+i)
    }
    printf("\n");
}
int main(void){
    int a[5] = {1,2,3,4,5};
        /*
    a[6] = {1,2,3,4,5,6};
    a[0]  a[1]  a[i]    i就是下标
    */
    //f(a, 1000);//下标越界
    int b[6] = {-1,-2,-3,-4,-5};
    int c[100] = {1,99,22,33};//C语言中如果不赋值是垃圾值，如果数组一半赋值一半未赋值，未赋值的是0。
    /*
        通过指针发送首元素地址和长度
        可以让f函数修改主函数a数组中所有元素的值
        将a数组的首地址和长度发送给f函数，在f函数中对p操作相当于在主函数中对a操作
    */
    f(a, 5);
    /*
        在f函数中可以获取a数组的任何一个元素的内容获取到也可以把a数组的任何一个变量的值改写;
        在f函数中对p[i]操作相当于在主函数中对a[i]操作
    */
    return 0;
}
```

指针变量的运算

指针变量不能相加 不能相乘 也不能相除

如果两个指针变量指向的是同一块连续空间中的不同存储单元,则这两个指针变量才可以相减。

一个指针变量到底占几个字节【非重点】

预备知识:

sizeof(数据类型)

功能: 返回值就是该数据类型所占的字节数

例子: sizeof(int) = 4 sizeof(char) = 1

sizeof(double) = 8

sizeof(变量名)

功能: 返回值是该变量所占的字节数

假设 p 指向 char 类型变量(1 个字节)

假设 q 指向 int 类型变量(4 个字节)

假设 r 指向 double 类型变量(8 个字节)

p q r 本身所占的字节数是否一样?

p q r 本身所占的字节数都是一样的。

总结:

一个指针变量, 无论它指向的变量占几个字节, 该指针变量本身只占 8 个字节(根据操作系统和编译器, 64 位占 8 个, 32 位占 4 个)。

一个变量的地址使用该变量首字节的地址来表示。

指针和二维数组

③、指针和函数

④、指针和结构体[*]

⑤、多级指针

[多级指针.cpp]多级指针的意义是体现跨函数使用内存

```
#include <stdio.h>
f(int ** q){//*q是p; **q是i;
    **q = 100;
}
g(void){
    int i = 10;
    int * p = &i;
    f(&p);
    printf("%d\n", i);
}
int main(void){
    int i = 10;
    int * p = &i;//p是个指针变量,当然可以使用别的变量来保存这个变量的地址
    int ** q = &p;
    int *** k = &q;
    //k = &p;//error 因为k是int ***类型,k只能存放int **类型变量的地址
    printf("%d\n", ***k);//*k=q; **k=p; ***k=i;
    g();
    return 0;
}
/*
在dev c++中的输出结果是:
------------------------
10
100
------------------------
*/
```

2：
```
void f(void){
        Int  a[5] = {1,2,3,4,5};
        //20 个字节，程序员无法手动编程释放它它只能在本函数运行完毕时由系统自动释放
g(a, 5);
printf("%d\n",a[2]);
}
```

专题【动态内存分配】【重点难点】

传统数组的缺点：

1. 数组长度必须事先制定，且数组长度只能是常整数，不能是变量；

例子：int a[5] = {1,2,3,4,5};//ok

int len = 5;int a[len];//error

2. 传统形式定义的数组，该数组的内存程序员无法手动释放（在一个函数运行期间，系统为该函数中数组分配的空间会一直存在，直到该函数运行完毕，数组的空间才会被系统释放）等价于（数组一旦定义，系统为该数组分配的空间就会一直存在，除非该数组所在的函数运行结束）；

3. 数组的长度一旦定义，其长度就不能再被更改（数组的长度不能在运行的过程中动态的扩充或缩小）；

4. 传统方式定义的数组不能跨函数使用（A 函数定义的数组，在 A 函数运行期间可以被其他函数使用，但 A 函数运行完毕后，A 函数中的数组将无法被其他函数使用。）

为什么需要动态分配内存

动态数组很好的解决了传统数组的这 4 个缺陷；

传统数组也叫静态数组。

动态内存分配举例_动态数组的构造[难]

[malloc 的使用.cpp]（知识点：先行知识点）

```
/*
    2021年1月4日
    malloc 是 memory(内存) allocate(分配)的缩写
    请求分配一块内存
*/
#include <stdio.h>
#include <malloc.h> //不能省略
int main (void){
    int i;//分配了4个字节。(静态分配)    一般动态分配是使用系统规定的动态分配函数实现的
    int  * p = (int *)malloc(4);//(int *)是强制类型转换,把第一个字节的地址强制转换成存放整型类型地址的类型
                                //强制类型转换告诉别人第一个字节地址到底是什么类型的地址
                                /*
                                4表示分配4个字节，而malloc只能返回第一个字节的地址，所以需要一个强制类型转换，
                                来告诉别人：第一个字节的地址指向的变量占几个字节，int *表示占4个字节 。
                                */
                        /*
                            1.要使用malloc函数，必须添加<malloc.h>这个头文件;
                            2.malloc函数只有1个形参，并且形参是整型(int和long int);形参就是需要分配的字节的个数
                            3.malloc(4)：4表示请求系统为本程序分配4个字节;
                            4.malloc函数只能返回第一个字节的地址.
                            5.10行分配了8个字节，p占4个字节，p所指向的内存也占4个字节
                            6.p本身所占的内存是静态的，而p所指向的内存是动态分配的

                        */
    *p = 5;//*p代表的就是一个int变量，只不过*p这个整型变量的内存分配方式和9行的i变量的分配方式不同
    free(p);//free(p);表示把p所指向的内存给释放掉
            /*
                p本身是静态的，不能由程序员手动释放，p本身的内存只能在p变量所在的函数运行中只是由系统释放。
                p所指向的是动态分配的内存
            */
    printf("同志们好！\n");
    return 0;
}
/*
    如果是  数据类型 变量名 = 值;就是静态分配;
    如果是  使用了malloc函数，后面加(形参)；就是动态分配.
*/
```

[动态一维数组示例_1.cpp]

```
/*
    体现出来的优点(只体现出3个,还有1个没体现出来):
        1、长度不需要事先指定;
        2、内存可以由程序员手动分配、手动释放;
        3、动态分配的内存可以在程序运行过程中,动态的增加或动态的减少.
    没有体现出动态内存可以跨函数使用(4)
*/
#include <stdio.h>
#include <malloc.h>
int main (void) {
    int a[5];//如果int占4个字节,则本数组包含了20个字节, 每4个字节被当作一个int变量来使用
    int len;
    int * p;
    int i;
    //动态的构造一维数组
    printf("请输入你需要存放的元素的个数");
    scanf("%d", &len);
    p = (int *)malloc(4 * len);//类似于:int p[len];(动态的构造了一个一维数组,数组名是p,数组长度是len,数组的每个元素类型是int类型)

    //对一维数组进行操作
    //对动态一维数组进行赋值
    for(i=0; i<len; i++)
        scanf("%d", &p[i]);

    //对一维数组进行输出
    for(i=0; i<len; i++)
        printf("p[i]=%d", p[i]);
    free(p);//释放掉动态分配的数组
    return 0;
}
/*
    4行的a数组与10行的p数组
    从内部而言,两个数组不一样,一个是静态的,一个是动态的
    在使用方面,可以把两个数组看成一个数组,(可以写成a[0]p[0]~~~)
*/
```

静态内存和动态内存的比较

静态内存是由系统自动分配,由系统自动释放;

静态内存是在栈分配的;

动态内存是由程序员手动分配,手动释放;

动态内存是在堆分配的(是以堆排序的方式分配的).

跨函数使用内存的问题

[跨函数使用内存_1. cpp]

//静态内存跨函数使用

```
#include <stdio.h>
f(int ** q){//接收指针变量的地址,形参的类型必须是(int **)类型
    int i = 10;
    //*q = i;//*q=p;     等价于p=i;
    *q = &i;//*q==p; 语句的含义就是将i的地址发送给p.
}
int main(void)
{
    int * p;
    f(&p);  //使用一次f函数后,p变量内保存的是i的地址.
            //要想改变p的地址,只能发送p的地址,所以f函数接收时形参的类型必须是(int **)类型
    printf("%d", *p);//本语句语法上没有问题, 但逻辑上有问题(访问了一个没有权限访问的内存,这种问题只有在指针中才会出现).

    return 0;
}
/*
    在10行调用f函数,当函数执行完后,在f函数执行中划分的变量i的地址已经被释放,在第12行中已经不可以访问了,但是由于软件太弱智,
    所以对这个越界行为没有报错. (函数执行完成,分配的内存释放,p中可以保存i的地址,但是不可以进行读取和改写,否则就是越界.)
*/
```

[动态内存跨函数使用示例_1.cpp]

```c
#include <stdio.h>
#include <malloc.h>
f(int ** q) {
    //如果需要代表p,则需要使用*q
    *q = (int *)malloc(sizeof(int));//sizeof(数据类型)表示该数据类型变量所占的字节
    //等价于 p = (int *)malloc(sizeof(int));
    //*q = 5;//等价于p = 5;
    **q = 5;//等价于*p = 5;
}
int main (void) {
    int * p;
    f(&p);
    printf("%d\n", *p);
    return 0;
}
```