#include<stdlib.h>

define 宏定义 （宏替换）

语法：

#define 宏名 字符串 //不带参数的宏

#define 宏名(参数) 字符串 //带参数的宏

功能：用指定的宏名（标识符）替代字符串

例如： #define n 10 n是符号常量（宏名）

说明：宏名**建议**用大写字母，变量**建议**小写字母

#define s(x) x+x

例题：

|  |
| --- |
| #define pi 3.14 void main() {float r=2.5,s; s=pi\*r\*r; printf(“%f”,s);} |

执行：宏定义时，用宏名代替字符串;编译前，用字符串替换掉宏名;运行时做运算;

例题：带参数的宏

|  |
| --- |
| #define s(x) x\*x void main() {int a=1,b=2; printf(“%d”,s(a+b)); //a+b\*a+b 输出5 } |

执行：定义时，用宏名代替字符串;编译前，用字符串替换宏名;如果含有参数，则要进行参数的替换;运行时做运算;

宏替换与函数调用的区别

1. 执行的时间不同，在编译前被执行的; 是在运行时调用的;

2. 处理方式不同，完成的是字符串的替换，不做运算; 先要计算出实参，然后调用，进行参数传递。

3. 占用内存空间不同，宏替换不占用内存空间; 函数调用形参也开辟内存空间

例题：有括号要带括号

|  |
| --- |
| #define min(x,y) (x)<(y)?(x):(y) void main() {int i=10,j=15,k; k=10\*min(i,j); printf(“%d”,k); } |

替换后：k=10\*min(i,j); 10\*(i)<(j)?(i):(j) ; // 运行时输出 15

**指针**

int a=2; 定义一个整型变量a，含义：是去内存开了一个int大小的房间，房间门上贴了标签a，房间里放了值2;

房间中的值共有2种存取方法：

1. 直接存取法：就是使用变量名（标识符）进行存或取;

2. 间接存取法：就是使用地址值进行存或取;

例如：void main(){int a=1,b=2,c; c=a +b; printf(“%d”,c); } //直接存取法

理解几个概念：

1. 地址值：内存房间的门牌号; 201

2. 指针：就是内存地址值的别名（第二个称呼）

3. 指针变量：是专门用来存放地址值的一种变量; 指针变量里面存储了指针（地址值）;

内存的地址是对外不可见的，所以看不到具体的地址值; 因此我们需要用&取地址运算符，获取地址值;

获取到的地址，需要存于指针变量中

4. 地址值是有类型之分，因此指针变量也有类型; 只有同类型的指针或地址，才能相互赋值或其他运算。

例如：int a=1; float b=2.5; &b此时的地址值是float型，地址的类型是由房间的类型决定的。

**指针变量的定义：需要使用 \* 运算符**

语法：类型 \* 指针变量名;

例如：int \* p; //定义了一个指针变量p，他的类型是 int \*;

p=地址值; p=&a; 是正确的 p=20;错误的

**给指针变量赋值：一定要使用地址值**

即：p=地址值（同类型的地址值）;

例如：p=&a; 约定：给指针变量赋值的操作，我们约定为指针变量的指向;

**利用指针变量操作房间中的内容（间接存取法）**; 语法: \*地址值：表示房间的内容

|  |
| --- |
| void main() {int a=1,b=2,c;  int \*p1,\*p2; //定义了两个指针变量;  p1=&a; p2=&b; //给指针变量赋值，约定指针变量完成了指向;  c= \*p1 + \*p2; // 利用指针变量操作房间中的值 printf(“%d”,c); } |

总结：使用指针三步骤：（使用指针变量操作内存房间中的值）

1. 定义： float \* p;

2. 赋值：p = 地址值; float型的地址

3. 使用：\*p;

说明：指针变量只负责操作房间中的值，不负责存储房间中的值。

|  |
| --- |
| int \*p; 定义  \*p=1; ERROR 缺少第二步;  int a=2; p=&a; 给指针变量赋值，约定为指针变量的指向; \*p=1; printf(“%d”,a); //输出1 |

例题：

|  |
| --- |
| void main() {int a =3,b=4,t; int \*p1,\*p2;  p1=&a; p2=&b; if(\*p1<\*p2) {t=\*p1; \*p1=\*p2; \*p2=t; }printf(“%d%d”,a,b); } //输出4 3 |

例题：

|  |
| --- |
| void main() {int a=2; int \*p =&a; 定义同时进行初始化赋值 \*p=3; printf(“%d”,a); //输出3} |

例题：

|  |
| --- |
| void main(){int a=3.b=4; int \*p1,\*p2,\*t; p1=&a; //p1=a; error p2=&b;  if(\*p1 < \*p2) { t= p1; p1=p2; p2=t;} printf(“%d%d”,a,b); //输出 3 4} |

总结：二者交换了地址值，实质是完成了指针的指向。

按数据类型来分类：

分为：整型的指针变量 int \* 浮点型指针变量 float \* 字符型指针变量 char \*

**\*按功能来分类：**

1.指向单个变量（单个房间的）指针变量; int a=2; int \*p float=2.5; float \* p;

2.指向数组元素的（每个元素都是单个房间）指针变量：这种类型就是1类型

3.指向整个一维数组的指针变量：

4.指向数组;是多个指针变量的集合：

5.指向函数的指针变量

6.函数的返回值类型是指针类型

用指针变量做函数的行参：

例如:

|  |
| --- |
| void swap(int a,int b){int t; t=a; a=b; b=t; }  void main(){ int a=3,b=4; swap(a,b); printf(“%d%d”,a,b);} |

对照：

|  |
| --- |
| void swap(int \*a,int \*b){int t; t=\*a; \*a=\*b; \*b=t;}  void main(){ int a=3,b=4; swap(&a,&b); printf(“%d%d”,a,b); } |

**参数传递规则1**;

如果函数的行参是基本数据类型变量（整型，字符，浮点）和结构体类型变量，则实参传递给形参，传递的是实参的拷贝值，行参与实参互不影响。

**参数传递规则2**;

如果函数的行参是数组或指针，则实参传递给形参，传递的是实参的地址值，形参会影响实参;

指向数组元素中的指针变量：与上述讲的指针变量相同;都是指向单个房间的

例如：a a+1 a+2 a+3 a+4 对应 int a[5] ={1,3,5,7,9};

int 型的5个元素;

int \* p; p=a+1; printf(“%d”,\*p); //输出 3

p = a; printf(“%d”,\*p); p=a+3; printf(“%d”,\*p) p++ 表示指针变量的指向，向后移动一个元素

例如：

p=a; for( ; p<=a+4; p++){printf(“%d”,\*p)}

总结：如果要指向单个房间的指针变量指向到了数组首元素上，则数组元素的引用，共有2类，4种方法：

1. 下标法：a[0] a[1] a[2] …. p[0] p[1] p[2] p[3]

2. 指针法（间接存取法）：\*p \*(p+1) \*(p+2) \*(p+3) \*a \*(a+1) \*(a+2) \*(a+3)

int a[5]={1,3,5,7,9}; int \*p； p=a;

|  |
| --- |
| void main() {int a[5]={1,3,5,7,9}; for(int i=0;i<5;i++) {if (a[i]%2 ==0) n++;}printf(“%d”,n); } |

对照： \*(p+i)%2==0 if(\*(a+i)%2==0) if(p[i]%2==0)

总结：

1. 允许指针变量指向到数组元素后，做++ - - 运算; 表示指针变量向后或向前 ，移动一个房间

2. 允许指针变量做关系运算，如：p<a+5; 此时按数组中的下标值，进行比较。

3. 允许指向同一个数组元素的两个指针变量，做减法 ; 表示这两个指针变量之间间隔的房间数。

4. 不允许指向同一个数组元素的两个指针变量，做加法。

5. 指针指向数组元素的指针变量做，\*p++运算，等价于\*(p++);

6. &与\*相遇，二者功能抵消; 例如 int a=2; int \*p=&a; printf(“%d”, \*&a); // 输出2

其他四类的指针变量

3. 指向整个一维数组的指针变量，与二维数组结合使用;

int(\*p)[m]; p是指针变量，指向含有m个元素的一维数组上;

int b[3][4]; 行地址 和 列地址;

int(\*p)[4];

p=b;