#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） float b=2.5; b房间称为目标变量

1. 定义指针变量： 类型\*指针变量名; 如： float \* p;

2. 给指针变量赋值：p = 地址值; float型的地址: p=&b;

3. 操作房间中的内容：\*地址值：如：\*p表示房间中的值2.5;

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=行地址; 即p=b; p指针指向了首行上; p指针也称为行指针;

引用元素的方法：行指针p前加\*,实现行指针转列指针; 列地址前加\*,表示内容\*(\*(p+1)+2) b[1][2]

4. **int \*p[4];** p是数组名，这是一个指针数组，说明数组中的每个元素都是一个指针变量，共有4个指针变量。

5. **int (\*p)();** 指向函数上的指针变量; p是指针变量 给p赋值时，需要使用函数名;

6. **int \*p();** 函数的返回值，是指针类型;

7. **int \*\*p;** 二级指针; 行指针; \*(\*p)

总结：指针是万能的，能操作所有的内容;

**字符型的指针变量;**

定义：char \* p; 定义了一个字符的指针变量，他即可以指向单个房间，也可以指向字符串;

例如：

char ch=’A’; char \*p=&ch; p指针变量指向到了房间ch上;

char \*st=”abc”; st指针变量指向到了字符串abc首元素上;

**等价与**：

char s[]=”abc”; char \*st; st=s; st是指向到了数组元素上的指针元素

**注意区分**：

char st[]=”abc”; ok char st[100];

st =”abc”; ERROR！ 原因是st表示常量地址值，=左边必须用常量

char \*s=”abc”; char \*s; s=”abc” 原因是s指针变量，是变量，可以放在=左侧。

删除算法：在一个串（数组）中删除一个指定的字符;

原理：要从前向后依次扫描每个字符，如果遇到的是要删除的字符，则不予处理，如果是遇到非删除字符，则向前放。

void fun(char \*s,char ch){ int i=0; while(s[i] != ‘\0’)

{ if(s[i]!=ch){ s[j]=s[i]; j++;} i++; } s[j]=’\0’; //有串必有\0 }

void main(){char st[]=”abbbcdbbc”; fun(st,’b’); puts(st); //abdc}

例题：void main(){char st[]=”how do you do” char \*p=st+strlen(st)/2; strcpy(p,”es it”); puts(st);}

//输出结果为 how does it

字符型的指针数组

main函数: void main() int main(){return 0;} int main(int argc,char \*argv[]){}

main函数中的形参含义：

1. argc表示命令行参数中，包括文件名在内的所有参数的总个数

2. 指针数组，数组中共含有argc个指针变量。每个指针变量都指向到包含文件名在内的所有参数上。

**结构体：**编程

应用情况：在程序中处理复杂数据时，需要使用结构体;

什么是复杂数据：数据本身含有多个属性或成分;

例如：int a=1,b=2; 简单的数据 int基本数据类型

生日 2000 01 02 | 书名，作者，单价，出版社 复杂的数据

如何使用结构体？

定义：结构体就是逻辑上相关联的一组数据的集合，这组数据可以是相同类型的，也可以是不同类型的。

定义结构体类型的语法： struct 结构体名 { 成员1; 成员2; …... }**;**

例如：

struct birth{int year; int month; int day;}; **说明**：定义了一个结构体类型

1. struct 是定义结构体类型的关键字

2. birth 是结构体名

3. struct birth 组合是结构体类型;

总结：结构体提供了一种给**用户自定义新的数据类型**的机会

利用结构体类型来定义结构体变量：

例如：int a,b,c; int是数据类型 abc是变量名 数据类型的作用是决定在内存中开辟多大空间的

struct birth a,b,c; 前者为数据类型 后者为结构体变量;

定义结构体变量共有3种方法：

1. 先定义结构体类型，然后再定义结构体变量(两件事分开)。

struct birth{int year;int month; int day;}; 第一件事：定义了结构体类型

struct birth a,b,c; 第二件事，定义了结构体变量 a,b,c

2. 定义结构体类型的同时，直接定义结构体变量(两件事同时做)。

struct birth{int year;int month; int day;}a,b,c;

3. 定义结构体类型的同时，直接定义结构体变量，此时可以省略结构体名;

struct{int year;int month;int day;}a,b,c;

给结构体变量进行初始化赋值：

例如：int a=2; 定义的同时初始化赋值

struct birth a={2000,1,2}; 说明：

1. 使用大括号，为类型中的每个成员进行赋值;

2. 赋值按成员的顺序一一赋值，中间的成员值不能省略; 后面的可以省略; 省略后默认是0;

总结：给所有的结构体变量进行赋值，一定就是给变量中的成员进行赋值

每个结构体变量都有属于自己的成员值;

例如：struct student{int id; char name[30]; float c1; float c2; };

struct student woNg={1,”王”,60,70};

给结构体变量，定义后再赋值;

例如： int a;定义一个整型变量 a

a=2; 给他赋值2

struct birth a,b;

a=2000; ERROR！ 切记：给结构体变量进行赋值，就是给其成员进行赋值

a.year=2000; a.month=1; a.day=2;

结构体变量**引用成员**的方法：使用 . 运算符; . 取成员运算符（起修饰限制作用）

例如：中国.辽宁.大连

例如：struct student woNg; 结构体变量赋值，不能直接给woNg赋值，需要给其成员进行赋值。

woNg.id=1; strcpy(woNg.name,”王”); woNg.c1=60;

例如：struct book {int id; char name[30]; float price;}

void main(){struct book java; scanf(“%d%s%f”,&java.id,&java.name,&java.price);

printf(“%d%s%f”,java.id,java.name,java.price); }

总结：结构体编程中的操作，永远要对结构体变量中的成员进行操作，不能直接对结构体变量进行操作。

结构体变量占据内存空间大小，等于所有成员占据内存空间之和;

sizeof(类型 变量); sizeof(struct book); 38 sizeof(java); 38

结构体类型的相互嵌套：就是在类型体内又使用了另一个结构体类型;

struct student{int id; char name[30]; struct birth{ int year; int month; int day;}birthday;};

等价与:

struct birth{int year; int month; int day;};

struct student{int id; char name[30]; struct birth birthday;}

使用：struct student guangqiang; sizeof(struct student); //4+30+12 个字节

guangqiang.id=2; strcpy(guangqiang.name,”光强”); guangqiang.birthday.year=2000;

说明：结构体嵌套类型变量，需要一级一级的引用到最底层的成员;

结构体类型的数组：

struct birth a[10];

结构体类型的数组,共有10个元素,每个元素都是结构体类型,即每个元素都包含3个成员;

例如：

struct book{int id; char name[30]; float price;}b[3]={{1,”cc”,30},{2,”java”,40},{3,”net”,50}};

printf(“%d%s%f”,b[0].id,b[0].name,b[0].price); 归属，限制。

结构体类型的指针：

struct book \*p; 定义了一个结构体类型的指针变量p

struct book cc; p=&cc; p指针指向了cc空间

p-> id p-> name p-> price 原因：结构体程序，永远要操作成员值，需要使用指针变量取成员值。

引用成员值，此时需要用-> 运算符。

总结：结构体变量下的成员，共有3种引用方法;

1. 如果是结构体变量，需要使用 . 运算符; cc.id , cc.name;

2. 如果是结构体指针变量，需要使用 ->运算符,p=&cc; 指针变量完成指向后p->id,p->name,p->price

3. 如果是结构体指针变量，可以先转换为结构体变量，然后再.运算符取成员 ，括号不能少！

(\*p).id cc.id (\*p).name cc.name

例如：

struct stu{int x; int y; int z;};

void main(){struct stu s[2]={{1,2,3},{4, 5,6}; struct stu \*p=s;

printf(“%d”,p- >y); printf(“%d”,++p- >x);} //输出 2 2;

\*例题：编程，输入5个学生的姓名和两门课成绩，计算每个学生的平均成绩，并输出姓名和平均成绩;

分析：学生：姓名，成绩1，成绩2; 复杂数据 使用结构体

结构体编程步骤:  
1.先定义结构体类型，定义时，要整合所有的相关属性或成分;

2.程序要永远针对成员进行操作，但一定要告诉每个成员的归属。

|  |
| --- |
| struct student { char name[30]; float c1; float c2; float avg;};  void main(){struct student stu[5];  for(int i=0;i<5;i++){  scanf(“%s%f%f”,&stu[i].name,&stu[i].c1,&stu[i].c2);  stu[i].avg=(stu[i].c1+stu[i].c2)/2;  printf(“%s%f”,stu[i].name,stu[i].avg); } } |

其他的几个关键字：

1. union 共用体，用于与结构体 struct 相同

用法： union 名{成员1; 成员2; };

union book{int a; int b;}; 共用体类型 union book b1,b2;

共用体变量占据内存空间大小，取决于变量中占内存空间最多的那个成员;

2. enum 枚举 一一列举;

枚举是有限个整形常量的集合; 与 struct 结构体相同

enum名{成员1,成员2,成员3};

enum color {red, green,blue,yellow};

常量值有两种：

1. 隐式常量值：第一个是0，依次递增

2. 显示常量值：用户自己定义的，未定义值的，以前一个为依据，依次递增。

enum color{red=10,green=100,blue,yellow=200};

enum color c1=red;

他们都是用户自定义，构造的类型。

4. typedef 关键字

作用：给数据类型起别名的

例如：typedef int aa; int k =2; 等价 aa k=2;

typedef 类型名 新名;

文件：磁盘文件

C语言中把磁盘文件当做了由一个个字符组成的序列;

分类：按数据在磁盘中的存储方式

1. 文本文件：数据按字符的ASCII值形式进行存储。

2. 二进制文件：数据在内存中的存放形式，直接存储到文件。

int a=12345;

文本: ‘1’’2’’3’’4’’5’ ‘49’ ‘50’ ‘51’ ‘52’ ‘53’ 5个字节

二进制：存数据的补码 4字节 文件中是数据的补码

区别：文本文件占空间大，读写速度慢，可以用文本编辑器，直接看原内容

二进制占空间小，读写速度快，不可以看文件原内容。

文件的操作方式：以内存为中心（你就是内存）

1.输入（读）：将数据拉到内存中的过程，称为输入

2.输出（写）：将数据送出去的过程，称为输出

读写方式分两种：

1. 顺序读写：逐个读写每个字符；更适合于文本文件

2. 随机读写：按指定的位置，读写指定大小，更适合于二进制文件，以完整的单位进行读写;

文件的操作步骤：

都是用函数来实现的；文件上的输入和输出函数，都在stdio.h文件中，使用之前必须添加。

操作步骤：

1. 打开文件 ： fopen，建立一条内存到文件之间的通道;

2. 读写文件内容：

3. 关闭文件： fclose，切断内存到文件之间的通道;

1. fopen() 函数

功能：打开文件，建立通道

语法：fopen(“文件名”,”文件的使用方式”);

文件打开成功后，会返回一个FILE \* 类型的指针

例如：

FILE \* fp; 文件指针

fp=fopen(“1.txt”,”w”); 打开文件1.txt ，成功打开后，fp指向到该文件上；

文件的使用方式：12种

1. 文本文件 二进制文件

"r" 只读 "rb"

"w" 只写 "wb"

"a" 只追加写 "ab"

"r+" 读写 "rb+"

"w+" 写读 "wb+"

"a+" 读追加写 "ab+"

+ 实现即可以读，又可以写

说明：

1. 带w字样的方式，如果文件不存在，会自动创建文件;

如果文件存在，文件的内容要先清空；只能先写再读。

2. 带r字样的方式，如果文件不存在，会报错;

如果文件存在，文件的内容不清空；可以先读，如果写，新内容覆盖原内容。

3. 带a字样，写是追加写; 如果文件不存在，会创建文件。

文件打开失败时会返回一个NULL（空指针）;

文件指针不负责文件中内容的操作，只负责文件属性的一些操作。

2. fclose()

功能：关闭文件，切断通道，如果未关闭，会造成数据的丢失

语法：fclose(文件指针);

例如：fclose(fp); 关闭fp指向的文件通道

3.文件内容的操作函数： 键盘和显示器的操作函数

(1)单个字符的读写函数 fgetc fputc getchar putchar

(2)字符串的读写函数 fgets fputs gets puts

(3)按指定的格式读写函数 fscanf fprintf scanf printf

(4)按数据块读写函数（二进制文件）fread fwrite

1. fgetc

功能：从文件中，读一个字符

语法：fgetc(文件指针);

例如：char ch; ch=fgetc(fp);

键盘：getchar char ch; ch=getchar();

2.fputc

功能：向文件中，写一个字符

语法：fputc(字符，文件指针);

例如：char ch=’A’; fputc(ch,fp);

显示器：putchar char ch=’A’; putchar(ch);

3.fgets

功能：从文件中，读一个字符串

语法：fgets(字符串，个数，文件指针); //最多读个数-1

例如：char st[100]; fgets(st,100,fp);

从fp指向的文件中，最多读100-1个字符，加上\0后，放入数组st中;

键盘：gets char st[100]; gets(st);

4.fputs()

功能：输出字符串到文件中

语法：fputs(字符串，文件指针)

例如：char st[]=”abc”; fputs(st,fp);

显示器：puts char st[]=”abc”; puts(st);

5.fscanf()

功能：从文件中，按指定的格式读数据

语法：fscanf(文件指针,“控制格式”,地址列表);

例如：int a; float b; fscanf(fp,”%d%f”,&a,&b);

键盘：scanf(“%d%f”,&a,&b);

6.fprintf()

功能：按指定的格式输出数据

语法：fprintf(文件指针,”控制格式”,输出列表)

例如：int a=2; float b=2.5; fprintf(fp,”%d%f”,a,b);

显示器：printf(“%d%f”,a,b);

7-8. fread fwrite 二进制文件

功能：读 写 数据块

例如： int a[]={1,3,5}; int b[3]; fwrite(a,sizeof(int),3,fp); fread(b,sizeof(int),3,fp);

a相当于缓存，sizeof(int)相当于每个元素大小，3相当于个数，fp相当于文件指针。

读写位置指针：用于控制文件内容操作的一种指针

feof:判断文件是否读到文件尾; 如果读到尾，返回1，没有读到尾，返回0;

while(!feof(fp)){ }

fseek:读写位置指针的移动函数

语法：fseek(文件指针,偏移量,起始位置标记)

起始位置标记

0 SEEK\_SET  
1 SEEK\_CUR

2 SEEK\_END

例如：fseek(fp,-10,2);

rewind:读写位置指针的回绕函数

语法：rewind(文件指针);

例题：实现文件的拷贝，复制 A.txt B.txt

|  |
| --- |
| #include<stdio.h>  void main()  { FILE \*fa,\*fb;  char ch;  fa=fopen(“A.txt”,”r”);  fb=fopen(“B.txt”,”w”);  while(feof(fa)!=1)  { ch=fgetc(fa);  fputc(ch,fb);  if(feof(fa) == 1) break; }  fclose(fa); fclose(fb); } |