

1.选择10x1.5

- ☒ 判断变量标识符
- ☒ 判断是否是字符常量
- ☐ 整型和十六进制的转换 %d %x

【例】输入一个十进制数，输出它对应的八进制数。

```
include<stdio.h>
int main(void)
{
    int x, i, n;
    int r[100];
    scanf("%d", &x);
    i=0;
    while(x)
    {
        r[i]=x%8;
        x=x/8; //重新计算x
        i++; //统计数组元素个数
    }
    n=i; //数组实际元素个数

    for(i=n-1; i>=0; i--)
        printf("%d", r[i]);
    printf("\n");
    return 0;
}
```

将余数依次存入数组中

逆序输出数组中的元素

18

- ☐ 表达式的值的分析
- ☒ 前置加和后置加的含义与二义性的规避

在一个表达式中不要多处出现变量的自增、自减等运算，同时避免与其它运算符连用
尽量不要在函数参数中用自增减表达式

自增、自减运算符只能用于变量，不可用于常量和表达式

因为表达式在内存内没有具体空间，常量所占的空间不能重新赋值

3++ (x+y)++ (-i)++ ✗

当出现同一级别的多个运算符时，按自右至左方式结合

$-i++ \Leftrightarrow -(i++)$

对于 $i+++j \Leftrightarrow (i++)+j$

$*p++ \Leftrightarrow *(p++)$

C语言解析时尽量往长记号进行解析，所以++一般处理成++而不是+ ++

若 $i=3, j=2$ $(i++)+j$ 等于 5 $i=4, j=2$

- ☒ 逗号表达式
- ☒ sizeof表达式的含义-常量表达式&无符号整型
- ☒ printf与scanf的用法与格式控制符

(注意：输入double型数据用%lf，而非%f)

h：加在 d、o、x、u 之前，表示输出 short

l：加在 d、o、x、u 之前，表示输出 long

L：加在 f、e、g 之前，表示输出 long double

- ☐ 函数调用的方式
- ☐ 指针作为函数参数的情况与数组变量的双向传递

指针参数应用

编写三个函数分别完成指定一维数组元素的数据输入、求一维数组的平均值、求一维数组的最大值和最小值。由主函数完成这些函数的调用。

分析：采用地址传递的方式，把一维数组的存储首地址作为实参数调用函数。在被调用的函数中，以指针变量作为形式参数接收数组的地址。该指针被赋予数组的地址之后，使用这个指针就可以对数组中的所有数据进行处理。

函数原型：

```
void input(float *,int);
float average(float *,int);
void maxmin(float *,int, float *,float *);
```

指针参数：地址传递

使函数送回多个值

最大值和最小值需要返回，采用地址传递方式

指针参数应用

```
#include <stdio.h>
void input(float *,int);
float average(float *,int);
void maxmin(float *,int, float *,float *);
int main()
{
    float data[10]; //一维数组定义
    float aver,max,min;
    float *p=data;
    input(data,10);
    aver=average(data,10);
    maxmin(p,10,&max,&min);
    printf("aver=%f\n",aver); //输出平均值
    printf("max=%f,min=%f\n",max,min);
    return;
}
float average(float *pdata,int n) //数组平均值函数
{
    int i; float avg;
    for(avg=0,i=0;i<n;i++) avg+=pdata[i];
    avg/=n; //求平均值
    return(avg); //将平均值返回给被调用函数
}
void input(float *pdata,int n) //输入数据函数
{
    int i;
    printf("please input array data: ");
    for(i=0;i<n;i++) //逐个输入数组的数据
        scanf("%f",&pdata[i]);
}
void maxmin(float *pdata,int n, float *pmax, float *pmin)
{
    int i;
    *pmax=*pmin=pdata[0];
    for(i=1;i<n;i++)
    {
        if(*pmax<pdata[i]) //求最大值
            *pmax=pdata[i];
        if(*pmin>pdata[i]) //求最小值
            *pmin=pdata[i];
    }
}
```

//形参可否改成 const float *pdata?

☒ 循环do-while

2. 填空2x6

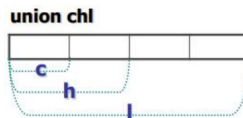
☐ Union联合体的各变量size

9.6 联合

- 与结构类似，联合类型也是一种构造类型。
- 结构变量的各成员变量占据各自不同空间。
- 联合变量的各成员变量占用同一内存空间。
- 联合特点：各成员共享存储

```
struct chl {
    char c;
    short h;
    long l;
};
```

```
union chl {
    char c;
    short h;
    long l;
};
```



联合存储区域的大小由各个成员中所占字节数最大的成员决定

一个联合类型中的所有成员共享同一个存储区域

联合变量的声明、初始化 和成员引用

- 除了用关键字union取代struct之外，联合类型的定义、联合变量的声明、以及联合成员的引用在语法上与结构完全相同。即可采用3种方式之一：

1 联合类型和其变量分别声明

```
union chl {
    char c;
    short h;
    long l;
};
union chl v = {.l=0x9876};
```

C99标准：可以对联合变量的任意成员初始化

```
76 98 00 00
v.c
v.h
v.l
printf("%c",v.c); ??
```

2 联合类型和其变量同时声明

```
union chl {
    char c;
    short h;
    long l;
} v = {.l=0x9876};
```

3 typedef定义之后声明变量

```
typedef union chl {
    char c;
    short h;
    long l;
} CHL;
CHL v = {.l=0x9876};
```

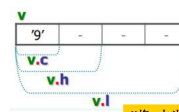
53

联合变量的声明、初始化 和成员引用

- 可采用3种方式之一：
 - 联合类型和其变量分别声明
 - 联合类型和其变量同时声明
 - typedef定义之后声明变量

早期C标准规定：只能对联合的第1个成员进行初始化。

```
union chl {
    char c;
    short h;
    long l;
} v = {'9'};
```



```
v.h=100;
v.l=0x9876;
printf("%c",v.c);
printf("%c",v.h);
```

//将v中当前的long值作为char和short来解释

联合所有成员都是从同一存储空间的边界（低地址）开始存放，联合各成员的地址和联合变量的地址是相同的，但类型不同

```
&v    &v.c    &v.h    &v.l    值相同
|      |      |      |
union chl *  char *  short *  long *
```

54

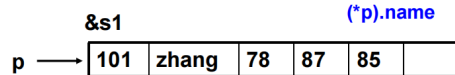
☐ 结构体中的指针变量指向另一个结构体/数组

3. 结构指针---指向结构变量的指针【9.4】

```
struct student s1 = {101, "zhang", 78, 87, 85}; //定义结构变量s1
```

```
struct student *p=&s1; //将结构指针p指向结构变量s1
```

从而，通过***p**可以间访**p**所指向的结构变量**s1**的成员内容，如，(***p**).num



结构指针的使用

(1)用 **->** 访问指针指向的结构成员。如:

```
printf("%d",p->num); strcpy(p->name,"Zhang");
```

(2) 用*p访问结构成员。如:

```
printf("%d", (*p).num);
```

注意: 括号不能少!

***p.num是非法表达式!**

下面三条语句等效:

```
s1.num = 101;
```

```
p->num = 101;
```

```
(*p).num = 101;
```

□ 位运算, 逻辑运算, 表达式的值, 运算优先级与结合性(与前后置运算符一起)

3.程序补全4x5

□ 课上例题

□ 数组逆序

将一个从键盘输入的整数序列按逆序重新存放并显示，整数个数由键盘输入；如输入5个数：8，6，5，4，1，则要求改为1，4，5，6，8保存。

```
#include <stdio.h>

int main ( )
{
    int a[100];
    int i, j, n, temp;
    printf("input the numbers:\n");
    scanf("%d", &n);
    for(i=0;i<n;i++) //输入整数序列
        scanf("%d",&a[i]);
    reverseA(a,n); //函数调用
    printA(a,n);
    return 0;
}
```

```
void reverseA(int a[ ], int n)
{
    int temp;
    for(int i=0, j=n-1; i<j; i++, j-- )
    {
        temp=a[j]; a[j]=a[i]; a[i]=temp;
    }
}

void printA(int a[ ], int n)
{
    for(int i=0; i<n; i++)
        printf("%5d",a[i]);
}
```

23

□ 整型转换字符串

整数转换为十进制数字串 (itoa)

转换之后的数字串怎么返回？函数不能直接返回数组类型

➤ 形式参数:

待转换的整数: n

转换结果串：字符数组s

➤ 返回值:

无

➤ 函数原型

```
void myitoa(int n, char s[]):
```

➤ **算法：**分解出n的每一位数字存于数组s中。

```
void myitoa(int n, char s[]);
```

算法思想:

n	n/10	n%10	s[]
1348	134	8	'8'
134	13	4	'4'
13	1	3	'3'
1	0	1	'1'

①计算n的模BASE之余数，转换成对应ASCII码，顺序存入字符数组

§:

②对 n/BASE 之商, 重复上述步骤

①，直到商等于0为止。

③逆置字符串s。

```
#define BASE 10
// converts a integer n to a string s
void my_itoa(int n, char s[]){
    int sign, j=0;
    if((sign=n<0) n=-n; // 处理负数
    while(n>0){
        s[j++] = n%BASE + '0' ;
        n /= BASE;
    }
    if(sign<0) s[j++] = '-'; // 处理负
    s[j] = '\0';
    mystrrrev(s);
}
```

108

109

□ 大小写转换(位运算)

【例2.21】字母的大小写转换

表达式: $c \wedge 32$

'A'	0100 0001	0x41	只需翻转第5
'a'	0110 0001	0x61	位, 掩码为 2^5

☐ 文件的读写(wb,w,r,rb.....)

fopen()

4.分析程序的输出结果6x4

☐ 链表排序

☒ 条件分支

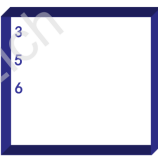
☒ switch-case执行

switch 语句

下面语句执行时的输出?

```
a = 0;
for(i=1;i<=3;i++){
    switch ( i ){
        case 0: a++;
        case 1: a++;
        case 2: a++;
        default: a++;
    }
    printf("%d\n", a);
}
```

case语句后无break; 程序依次执行后面的语句, 包括default语句



方案3—switch语句

```
float x;
printf("input the score x\n");
scanf("%f", &x);
if ( x > 100 || x < 0 ) printf("input error!\n");
else
    switch ( (int)(x/10) )
    {
        case 10:
        case 9: printf(" excellent! \n"); break;
        case 8: printf(" good! \n"); break;
        case 7:
        case 6: printf(" middle! \n"); break;
        default: printf(" bad! \n"); break;
    }
```

空的case语句, 用于几种情况合并执行一组语句。

☐ 简单的逻辑结构

☐ 循环体

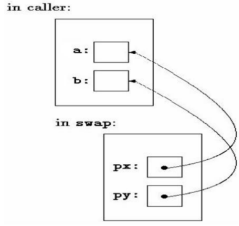
☐ 小指针作为函数参数

指针作函数参数

- 改变主调函数中变量的值
- 使函数送回多个值

传址调用: 以指针作为函数的参数实现变量值的改变。

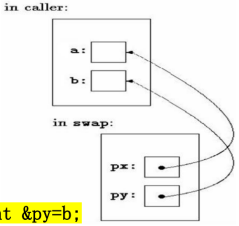
```
#include <stdio.h>
void swap(int *px, int *py)
{
    int t;
    t=*px; *px=*py; *py=t;
}
int main(void)
{
    int a=10, b=20;
    swap(&a, &b);
    printf("a=%d, b=%d\n", a, b);
    return 0;
}
```



引用作函数参数

传址调用: 以引用作为函数的参数实现数据的双向传递

```
#include <stdio.h>
void swap(int &px, int &py)
{
    int t;
    t=px; px=py; py=t;
}
int main(void)
{
    int a=10, b=20;
    swap(a, b); // int &px=a; int &py=b;
    printf("a=%d, b=%d\n", a, b);
    return 0;
}
```



☐ 整数转换为字符串

5.编程2x8+1x10

☐ 动态接收/创建数据(malloc函数)

重点

动态内存分配/释放

动态内存分配: 在程序运行期间动态地分配存储空间,分配的内存空间放在数据区的堆 (heap) 中

动态内存分配函数: `void * malloc(unsigned size);`

指定所分配内存空间的大小

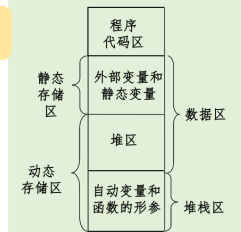
若分配成功,则返回被分配内存空间的首地址,否则,返回空指针NULL。当该内存不再使用时,应使用`free()`函数将内存释放。

动态内存释放:

`malloc()`函数在堆区中所分配的内存空间首地址

`void free(void * ptr);`

头文件: `#include <malloc.h>`或
`#include <stdlib.h>`



动态内存分配/释放

malloc函数调用一般形式

`#include <stdlib.h>`

`T *p=(T *)malloc(unsigned size)`

//T为指针的数据类型

`if(p == NULL) //`

`{`
 出错处理操作;
 `exit(1);`
`}`

`//.....`

`free(p);`

`p=NULL; //防止使用野指针`

- ☐ 递归函数(调用与出口)
- ☐ 排序(可能要求用指针实现)