

## 目录

1	引言 .....	3
1.1	obj 文件 .....	3
2	数据表达与组织 .....	3
2.1	常量, 变量, 运算与表达式 .....	3
2.1.1	C 语言中最简单的数据类型 .....	3
2.1.2	声明与定义 .....	3
2.1.3	运算符优先级表 .....	4
2.1.4	进制 .....	7
2.1.5	零和空 .....	8
2.1.6	变量大小 .....	8
2.1.7	位操作 .....	8
2.2	一维和二维数组, 字符数组和字符串 .....	9
2.2.1	定义二维数组 .....	9
2.2.2	注意: strlen() sizeof() 返回值是 unsigned int .....	10
2.2.3	字符串 .....	10
2.3	指针与数组, 结构与数组 .....	11
2.4	指针与结构, 单向链表 .....	12
3	语句及流程控制 .....	13
3.1	复合语句 .....	13
3.2	分支控制(if、switch) .....	13
3.3	循环控制(for、while、do—while) .....	13
4	程序结构和函数 .....	13
4.1	C 程序结构 .....	13
4.2	函数的定义、参数传递和调用 .....	13
4.3	函数的递归调用 .....	13
4.4	变量的存储类别、作用域, 全局变量和局部变量 .....	13
4.4.1	静态变量 .....	13
5	输入/输出和文件 .....	14

5.1	标准输入和输出 .....	14
5.2	文本文件与二进制文件 .....	14
5.3	文件打开、关闭、读写和定位 .....	14
5.3.1	函数 .....	14
6	编译预处理和命令行参数 .....	16
6.1	宏定义和宏函数 .....	16
6.2	命令行参数和使用 .....	16
7	基本算法设计与程序实现 .....	16
7.1	简单排序算法（插入、选择、冒泡）、二分查找（看《王道》） .....	16
7.2	链表、文件中查找 .....	18
7.3	级数求和、进制转换 .....	18
7.3.1	级数求和 .....	18
7.3.2	负数的进制转化变准输出 .....	18
7.3.3	进制转化例题 .....	19

浙大软院考研 QQ群: 100709798

# 1 引言

## 1.1 obj 文件

程序编译时生成的中间代码文件。目标文件，一般是程序编译后的二进制文件（不可直接运行），再通过链接器和资源文件链接就可执行文件了。OBJ 只给出了程序的相对地址，而可执行文件是绝对地址。

# 2 数据表达与组织

## 2.1 常量，变量，运算与表达式

### 2.1.1 C 语言中最简单的数据类型

整型、实型、字符型

### 2.1.2 声明与定义

①变量定义：用于为变量分配存储空间，还可为变量指定初始值。程序中，变量有且仅有一个定义。

②变量声明：用于向程序表明变量的类型和名字。

③定义也是声明：当定义变量时我们声明了它的类型和名字。

④extern 关键字：通过使用 extern 关键字声明变量名而不定义它。

1.定义也是声明，extern 声明不是定义，即不分配存储空间。extern 告诉编译器变量在其他地方定义了。

例如：extern int i; //声明，不是定义

int i; //声明，也是定义

2.如果声明有初始化式，就被当作定义，即使前面加了 `extern`。只有当 `extern` 声明位于函数外部时，才可以被初始化。

例如：`extern double pi=3.1416; //定义`

3.函数的声明和定义区别比较简单，带有`{}`的就是定义，否则就是声明。

例如：`extern double max(double d1,double d2); //声明`

4.除非有 `extern` 关键字，否则都是变量的定义。

例如：`extern int i; //声明`

`int i; //定义`

2.1.3 运算符优先级表

优先级	运算符	名称或含义	使用形式	结合方向	说明
1	[ ]	数组下标	数组名[常量表达式]	左到右	--
	( )	圆括号	(表达式) / 函数名(形参表)		--
	.	成员选择（对象）	对象.成员名		--
	->	成员选择（指针）	对象指针->成员名		--
2	-	负号运算符	-表达式	右到左	单目运算符
	~	按位取反运算符	~表达式		
	++	自增运算符	++变量名/变量名++		
	--	自减运算符	--变量名/变量名--		

	*	取值运算符	*指针变量			
	&	取地址运算符	&变量名			
	!	逻辑非运算符	!表达式			
	(类型)	强制类型转换	(数据类型)表达式			--
	sizeof	长度运算符	sizeof(表达式)			--
3	/	除	表达式/表达式	左到右	双目运算符	
	*	乘	表达式*表达式			
	%	余数（取模）	整型表达式%整型表达式			
4	+	加	表达式+表达式	左到右	双目运算符	
	-	减	表达式-表达式			
5	<<	左移	变量<<表达式	左到右	双目运算符	
	>>	右移	变量>>表达式			
6	>	大于	表达式>表达式	左到右	双目运算符	
	>=	大于等于	表达式>=表达式			
	<	小于	表达式<表达式			
	<=	小于等于	表达式<=表达式			
7	==	等于	表达式==表达式	左到右	双目运算符	
	!=	不等于	表达式!= 表达式			

8	&	按位与	表达式&表达式	左到右	双目运算符
9	^	按位异或	表达式^表达式	左到右	双目运算符
10		按位或	表达式 表达式	左到右	双目运算符
11	&&	逻辑与	表达式&&表达式	左到右	双目运算符
12		逻辑或	表达式  表达式	左到右	双目运算符
13	?:	条件运算符	表达式 1? 表达式 2: 表达式 3	右到左	三目运算符
14	=	赋值运算符	变量=表达式	右到左	--
	/=	除后赋值	变量/=表达式		--
	*=	乘后赋值	变量*=表达式		--
	%=	取模后赋值	变量%=表达式		--
	+=	加后赋值	变量+=表达式		--
	-=	减后赋值	变量-=表达式		--
	<<=	左移后赋值	变量<<=表达式		--

	>>=	右移后赋值	变量>>=表达式		--
	&=	按位与后赋值	变量&=表达式		--
	^=	按位异或后赋值	变量^=表达式		--
	=	按位或后赋值	变量 =表达式		--
15	,	逗号运算符	表达式,表达式,...	左到右	--

### 2.1.3.1赋值

8 进制	int k = 0 数值 ,0 是零, 不是 o
16 进制	int k = 0x 数值 ,0 是零, 不是 o

### 2.1.3.2标准输入输出

10 进制	%d
8 进制	%o ,o 是字母, 不是零
16 进制	%x
左对齐	如: %-8d 一共 8 位 (包括小数点), 左对齐, 如果本身位数超过 8, 就失效了。 不能在小数点右边使用
补零	如: %08d 一共 8 位, 不够 8 位左边补零 %04.08lf 小数点后一共保留 8 位, 不够 8 位右边补零。本身位数超过 8, 就 04 就失效了。
float , double 变量输出的结果	显示的都是保留到小数点后 6 位
int a=3 printf("%d %d %d",++a,++a,++a); =>6 6 6 printf("%d%d%d",a++,a++,a++) =>5 4 3	在前++运算, 是先变量自增, 再调用 printf 函数。 printf 函数运行是从右到左依次输出的。

### 2.1.4 零和空

零或空	数值
'0'	48
'\0'	0
NULL	0
EOF	-1

### 2.1.5 变量大小

```
sizeof(short)=2
sizeof(int)=4
sizeof(long)=4
sizeof(long long)=8
sizeof(char)=1
sizeof(float)=4
sizeof(double)=8
sizeof(void*)=4
```

### 2.1.6 位操作

优先级： 取反~ > 移位<<、>>> > 与& > 异或^ > 或	
^ 异或	两数不同为 1，两数同 0
~ 取反	
<<>> 移位	高效替换 n/=pow(2,x) ----> n >>= x n*=pow(2,x) ----> n <<= x
& 与	与 0 为 0，与 1 不变  制 a 的第 i 位为 0： a & ~(1<<i)  判断 a 的第 i 位是否为 0 if(a&(1<<i) ==0)  判断奇偶 if(n&1 == 0)
或	或 0 不变，或 1 为 1



### 2.1.6.1 计算十进制整数的二进制中的 1 的数目

```
#include <stdio.h>
int getbit( int a,int i)
{
    return a&(1<<i);// 只取数字 a 第 i 位上的值
}
int main()
{
    int n;
    while(scanf("%d",&n)!=EOF)
    {
        int i;
        int count=0;
        for(i=0;i<32;i++)//int 32 位
        {
            if(getbit(n,i)!=0) ++count;
        }
        printf("%d\n",count);
    }
    return 0;
}
```

### 2.1.6.2 交换两个数，不用其他变量

1、  
Left=left+right  
Right=left-right  
Left=left-right  
2、  
位操作(整数):  
Left=left ^ right  
Right=left ^ right (=left^right^right=left^0=left)  
Left= left ^ right (=left^right^left=left^left^right=0^right=right)

## 2.2 一维和二维数组，字符数组和字符串

### 2.2.1 定义二维数组

注意:

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

是错误的，因为，如后两列为 0，也可以写成：

`int a[2][4]={{1,2},{3,4}};`所以省略后编译器不知道是多少列

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

是正确的，编译器可以根据赋值来确定有多少行

## 2.2.2 注意：strlen() sizeof() 返回值是 unsigned int

```
int i;
char s[] = "123";
for(i=-1;i<strlen(s)-1;i++)
{
    printf("1");
}
```

不会打出任何东西，`strlen()-1` 类型是 `unsigned int`，“`i<strlen()-1`”会把 `i` 转换成 `unsigned int`。而 `i=-1` 是 `unsigned int` 中 最大值

## 2.2.3 字符串

### 2.2.3.1 字符串定义

1、

```
char a[]=""; sizeof(a)=1;
char b[]="\0" sizeof(b)=2;
```

编译器都会给它 加个 `'\0'`

2、

```
char p[] = "hello"; //存储在变量区
```

在变量区为 `char p[]` 开辟了内存，把 `"hello"` 赋值给它。

```
char * p = "hello"; //存储在常量区，只读不写
```

在常量区开辟内存存储字符串 `"hello"`，指针 `p` 指向它。

### 2.2.3.2函数

strcat	<div>strcat </div> <p>将两个char类型链接。</p> <pre>char d[20]="GoldenGlobal", char *s="View"; strcat(d,s);</pre> <p>结果放在d中</p> <pre>printf ("%s", d) ;</pre> <p>输出 d 为 GoldenGlobalView（中间无空格）</p> <p>d和s所指内存区域不可以重叠且d必须有足够的空间来容纳s的字符串。</p> <p>返回指向d的指针。</p>
puts("ddqdw")	自动换行
gets(char *)	必须先对字符数组指针赋内存，读一行。不会检查数组越界，以“回车”键为止
strcpy 的 bug	<pre>void strcpy(char* dest, char* src) {     while(*dest++ = *src++); }</pre> <p>如果 strcpy (src+strlen (src), src) —— ‘\0’ 被覆盖了，则会出现 bug</p>

## 2.3 指针与数组，结构与数组

指针数组	如：int* a[3]
数组指针	如：int (*a)[3]  int arr[3][3];  int** a = arr;//出错!!! a 是指向（int*）的指针，而 arr 是指向第一行数组的数组指针

	<code>int (*a)[3] = arr; //正确</code> 此时: <code>* ( (* (a+1) ) +2)</code> 为数组第二行第三列的值
函数指针	返回值类型 ( <code>*p</code> ) (参数类型) = &函数名 如: <code>int fun (float a , char b) {}</code> <code>int (*p) (float , char) = &amp;fun;</code>
指针相减	<code>int* p1 ; int* p2;</code> <code>p1 - p2 = ((int)p1 - (int)p2) / sizeof(int*) = 数据项编号相减</code>

## 2.4 指针与结构，单向链表

注意：C 中定义结构体变量的时候，`struct` 不能省略  
如：

```
struct Node
{
    int val;
    struct Node* p; // struct 不能省略
};
struct Node* Link = NULL;
```

若想省略 可以用类型定义 `typedef`

```
typedef struct Node
{
    int val;
    struct Node* p;
} node;
node* Link = NULL;
```

注意 `typedef` 用法是 `typedef 类型名 1 类型名 2;`  
区别与

```
struct Node
{
    int val;
    struct Node* p;
} node;
```

这里

`struct Node{int val; struct Node* p; }` 是类型名  
`node` 是变量名

## 3 语句及流程控制

### 3.1 复合语句

### 3.2 分支控制(if、switch)

### 3.3 循环控制(for、while、do—while)

## 4 程序结构和函数

### 4.1 C 程序结构

### 4.2 函数的定义、参数传递和调用

### 4.3 函数的递归调用

### 4.4 变量的存储类别、作用域，全局变量和局部变量

#### 4.4.1 静态变量

外部静态变量	该变量只能在这个文件里面使用
内部静态变量	<p>不管函数是否被调用，一直存在。当函数多次调用时，不用再次定义：</p> <pre>void fun() {     static int b = 0;     ++b; }</pre> <p>多次调用 <b>fun()</b>，<b>b</b> 的值会不断增加，而不是每次先变成 <b>0</b>，再自增。</p>

## 5 输入/输出和文件

### 5.1 标准输入和输出

### 5.2 文本文件与二进制文件

### 5.3 文件打开、关闭、读写和定位

#### 5.3.1 函数

函数	功能	用法
<b>fopen</b>	打开文件	<p>FILE * fopen(const char * path, const char * mode);</p> <p>返回值：文件顺利打开后，指向该流的文件指针就会被返回。如果文件打开失败则返回 NULL。：</p> <p>参数 path 字符串包含欲打开的文件路径及文件名，参数 mode 字符串则代表着流形态。</p> <p>mode 有下列几种形态字符串：</p> <p>r 以只读方式打开文件，<b>该文件必须存在。</b></p> <p>r+ 以可读写方式打开文件，<b>该文件必须存在。</b></p> <p>rb+ 读写打开一个二进制文件，允许读写数据，<b>文件必须存在。</b></p> <p>w 打开只写文件，<b>若文件存在则文件长度清为 0，即该文件内容会消失。若文件不存在则建立该文件。注意：在 windows 下，w 模式 fputc('\n', fp)，写入文本的会是 '\r' '\n' 两个字节，wb 模式则没有这问题。</b></p> <p>w+ 打开可读写文件，<b>若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件。</b></p> <p>a 以附加的方式打开只写文件。<b>若文件不存在，则会建立该文件</b>，如果文件存在，写入的数据会被加到文件尾，即文件原先的内容会被保留。（EOF 符保留）</p> <p>a+ 以附加方式打开可读写的文件。<b>若文件不存在，则会建立该文件，如果文件存在</b>，写入的数据会被加到文件尾后，即文件原先的内容会被保留。（原来的 EOF 符不保留）</p> <p>wb 只写打开或新建一个<b>二进制</b>文件；只允许写数</p>

		<p>据。</p> <p>wb+ 读写打开或建立一个<b>二进制</b>文件，允许读和写。</p> <p>ab+ 读写打开一个<b>二进制文件，允许读或在文件末追加数据。</b></p>
<b>fprintf</b>	格式化输出到一个流/文件中， <b>在文件中以二进制存储</b>	<pre>int fprintf (FILE* stream, const char*format, [argument])</pre> <p>FILE*stream: 文件指针</p> <p>const char*format: 输出格式</p> <p>[argument]: 附加参数列表</p>
<b>fwrite</b>	向文件写入一个数据块	<pre>size_t fwrite(const void*buffer, size_t size, size_t count, FILE* stream);</pre> <p>注意：这个函数以二进制形式对文件进行操作，不局限于文本文件</p> <p>返回值：返回实际写入的数据块数目</p> <p>(1) buffer: 是一个指针，对 fwrite 来说，是要获取数据的地址；</p> <p>(2) size: 要写入内容的单字节数；</p> <p>(3) count: 要进行写入 size 字节的数据项的个数；</p> <p>(4) stream: 目标文件指针；</p> <p>(5) 返回实际写入的数据项个数 count。</p>
<b>fgetc</b>	从流中读取字符	<pre>int fgetc(FILE *stream);</pre> <p>这个函数的返回值，是返回所读取的一个字节。如果读到文件末尾或者读取出错时返回 EOF。</p>
<b>fseek</b>	重定位流(数据流/文件)上的文件内部位置指针	<pre>int fseek(FILE *stream, long offset, int fromwhere);</pre> <p>函数设置文件指针 stream 的位置。如果执行成功，stream 将指向以 fromwhere (偏移起始位置：<b>文件头 0(SEEK_SET)</b>，当前位置 1(SEEK_CUR)，文件尾 2(SEEK_END)) 为基准，偏移 offset (指针偏移量) 个字节的位置。如果执行失败(比如 offset 超过文件自身大小)，则不改变 stream 指向的位置。</p>
<b>rewind</b>	将文件内部的位置指针重新指向一个流(数据流/文件)的开头	<p><b>rewind 函数作用等同于 (void)fseek(stream, 0L, SEEK_SET);</b></p> <pre>void rewind(FILE *stream);</pre>
<b>ftell</b>	函数 ftell 用于得到文件位置指针当前位置相对于文件首的偏移字节数。	<pre>long ftell(FILE *stream);</pre>
<b>feof</b>	检测流上的文件结束符	<pre>int feof(FILE *stream);</pre> <p>如果文件结束，则返回非 0 值，否则返回 0</p>

## 6 编译预处理和命令行参数

### 6.1 宏定义和宏函数

### 6.2 命令行参数和使用

```
Int main(int argc ,char *argv[])
```

```
{
```

argc 参数的数量

argv 每个字符串对应一各参数

argv[0] 启动该程序的文件名，所以 argc 至少为 1

## 7 基本算法设计与程序实现

### 7.1 简单排序算法（插入、选择、冒泡）、二分查找

#### 7.1.1 插入排序

```
void Insert_Sort(int ans[],int n)
{
    for(int i = 1 ; i < n ; ++i)
    {
        int tem = ans[i];
        int j = i-1 ;
        while(j >= 0 && ans[j] > tem)
        {
            ans[j+1] = ans[j];
            --j;
        }
        ans[j+1] = tem;
    }
}
```

#### 7.1.2 选择排序

```
void Select_Sort(int ans[],int n)
{
```



```

for(int i = 0 ; i < n ; ++i)
{
    int MIN = i;
    for(int j = i+1 ; j < n ; ++ j)
    {
        if(ans[j] < ans[MIN])
        {
            MIN = j;
        }
    }
    if(MIN != i)
    {
        int tem = ans[i];
        ans[i] = ans[MIN];
        ans[MIN] = tem;
    }
}
}

```

### 7.1.3 冒泡排序

```

void Bubble_Sort(int ans[], int n)
{
    for(int i = n-1 ; i > 0 ; --i)
    {
        bool change = false;
        for(int j = 0 ; j < i ; ++ j)
        {
            if(ans[j] > ans[j+1])
            {
                change = true;
                int tem = ans[j];
                ans[j] = ans[j+1];
                ans[j+1] = tem;
            }
        }
        if(!change)
        {
            break;
        }
    }
}

```

### 7.1.4 二分查找

```
int Binary_Search(int ans[], int n, int key)
{
    int left = 0;
    int right = n-1;
    while(left <= right)
    {
        int mid = (left + right)/2;
        if(ans[mid] > key)
        {
            right = mid - 1;
        }
        else if (ans[mid] < key)
        {
            left = mid + 1;
        }
        else
        {
            return mid;
        }
    }
    return -1;
}
```

## 7.2 链表、文件中查找

链表题目合集: <http://www.cnblogs.com/xiaoyesoso/p/4234443.html>

## 7.3 级数求和、进制转换（考前背代码，做题）

### 7.3.1 级数求和

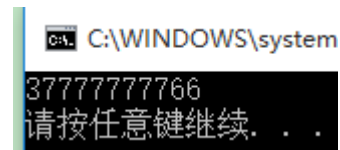
如果给出精确度  $x$ ，跳出循环的条件是，这次循环加上的项值  $\leq x$ 。

### 7.3.2 负数的进制转化变准输出

**C 语言中，进制转换的标准输出**

```
int main(void)
{
    int n = -10;
    printf("%o\n", n);
    return 0;
}
```

结果为:



```
C:\WINDOWS\system
3777777766
请按任意键继续. . .
```

自己编写模拟，只需 **n** 类型变为 **unsigned int**，再进行。

### 7.3.3 进制转化例题

<http://ac.jobdu.com/problem.php?pid=1080>

浙大软院考研 QQ 群: 100709798