联航训练营实训手册

C与数据结构

联航嵌入式精英训练营

文档信息

文档名称：	联航训练营实验手册		
版本号：	1.00	版本日期：	2013-05-8
质量审察阶段：	N/A		
编写者：	Tony Zhang	准备日期：	
审查者：		审查日期：	

更新说明

日期	更新内容

目 录

第一部分 C语言 5
1 C语言基础 5
1.1 第一个C语言程序 5
1.2 自定义函数调用与返回值 5
1.3 编程风格 5
2 变量与常量 6
2.1 变量的存储位置 6
2.2 类型提升 6
3 C程序控制 6
3.1 帮助老师为学生考试成绩排等级: 7
3.2 计算π的值： 8
3.3 请通过程序统计 1~100 中数字 9 的个数 8
3.4 编程求N的阶乘(N!) 9
3.5 编程求 100 以内的素数，并打印 10
3.6 求完数 11
3.7 打印所有的水仙花数 13
4 函数与递归 14
4.1 递归求n的阶乘（n!） 14
4.2 求Fibonacci （递归法） 14
4.3 最大公约数（递归与迭代） 15
4.4 字符串反转（递归法） 16
4.5 二分法查找(递归法) 16
4.6 汉诺塔 17
5 数组 18
5.1 冒泡排序 18
5.2 二分法查找(迭代法) 19
5.3 memcpy 20
5.4 矩阵转置 21
6 指针 22
6.1 命令行参数 22
6.2 指向指针的指针 22
6.3 函数指针 22
7 字符串 22
7.1 strlen 22
7.2 strcpy与strncpy 22
7.3 atoi 23
7.4 atof 24
7.5 strcmp与strncmp 26
7.6 strstr 27
7.7 reverse 28
7.8 strcat与strncat 28
7.9 判断是字符串是否回文 29

8 位运算.....30
8.1 以二进制打印无符号数.....30
8.2 统计一个无符号整数(unsigned int)的二进制表示中 1 的个数.....31
8.3 循环右移.....31
9 结构体、共用体、枚举与位字段.....32
10 预编译与宏.....32
11 综合训练.....32
11.1 约瑟夫环(使用数组).....32
11.2 约瑟夫环（使用环形链表）.....34
11.3 银行卡号校验.....36
11.4 单词逆序.....38
11.5 进制转换.....40
第二部分 数据结构.....42
1 线性表.....42
1.1 单向链表.....42
1.2 双向链表.....46
1.3 循环链表.....50
2 栈.....54
2.1 用链表实现栈.....54
2.2 用数组实现栈.....57
2.3 用栈走迷宫（深度优先）.....58
3 队列.....61
3.1 用链表实现队列.....61
3.2 用队列走迷宫（广度优先）.....65
4 二叉树.....67
5 查找.....71
6 排序.....71
5.1 冒泡排序.....71
5.2 插入排序.....71
5.3 选择排序.....72
5.4 shell排序.....73
5.5 快速排序.....74

第一部分 C 语言

1 C 语言基础

1.1 第一个 C 语言程序

```
#include <stdio.h>

int main(void)
{
    printf("Helloworld\n");
    return 0;
}
```

重点：（1）#include <stdio.h> 和 #include “stdio.h” 的区别。 ★头文件查找方式不同。

（2）库函数printf的参数有什么要求？返回值是什么？ 为什么不判断或保存返回值？

★返回成功打印的字符个数。

（3）return语句没有任何返回值会发生什么？返回 0 和非 0 值表示什么？

1.2 自定义函数调用与返回值

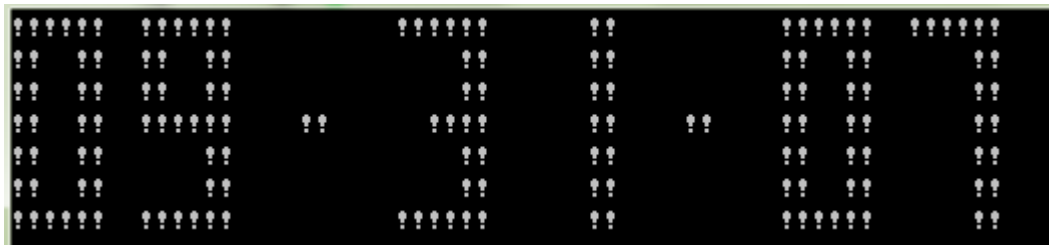
```
#include <stdio.h>
int add(int a,int b)
{
    int total ;
    total = a + b;
    return total;
}

int main(void)
{
    printf("total is:%d\n",add(3,5));
    return 0;
}
```

1.3 编程风格

下面这段代码即为第 19 届 IOCCC（国际混乱C语言代码大赛）优胜作品：“A clock in one line”。

```
main(_){_^448&&main(-~_);putchar(--_%64?32|-~7[__TIME__-_/8%8]["'>'txiZ^(~z?"-48]>>"";;;
====~$::199"[_*2&8|_/64]/(_&2?1:8)%8&1:10);}
```



2 变量与常量

2.1 变量的存储位置

```
#include <stdio.h>
int g_init = 100;
int g_uninit;
int main()
{

	int l_v;
	static int sl_init = 100;
	static int sl_uninit;
	printf("g_init :	%p\n",&g_init);
	printf("g_uninit :	%p\n",&g_uninit);
	printf("l_v :		%p\n",&l_v);
	printf("sl_init :	%p\n",&sl_init);
	printf("sl_uninit :	%p\n",&sl_uninit);
	printf("fun :		%p\n",&main);
	printf("string :	%p\n","helloworld");

	return 0;
}
```

2.2 类型提升

```
#include <stdio.h>
int main()
{
	unsigned char c1 = 255, c2 = 2;
	int n = c1 + c2;
	printf("%d\n",n);
}
```

3 C 程序控制

3.1 帮助老师为学生考试成绩排等级:

【90 分，100 分】为A，【80 分，90 分）为B，【70 分， 80 分）为C， 【60 分， 70 分）为D， （60 分， 0 分】为E。

```
#include <stdio.h>

char grade(float score)
{
	char ch;
	switch((int) score / 10)
	{
		case 10:
		case 9:	ch = 'A';
			break;
		case 8:	ch = 'B';
			break;
		case 7:	ch = 'C';
			break;
		case 6:	ch = 'D';
			break;
		default: ch = 'E';
			break;
	}

	return ch;
}

int main()
{
	printf("%c\n",grade(97));
	return 0;
}
```

重点：(1) 程序第 7 行为什么进行强制类型转换？ 以及除以 10 的意义何在？

★为了得到 0 至 10 之间的整数。

（2）第 9 行到第 10 行是否有错？为什么？ ★没错。

（3）第 16 和 17 行可以与第 19 和 20 行交换吗？ ★完全可以。

（4）default标号可以不写在最后吗？以及其后的break可以省略吗？

★可以不写在最后。当default不写在最后时不能省略break.

3.2 计算π的值：

根据下面的公式计算π的值

π = 4 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + …

请打印出一个表格来显示：用公式中的 1 项、2 项、3 项…计算出来的π的近似值。

```
#include <stdio.h>

double get_pi(int n)
{
	double total = 0;
	int i = 0;
	int flag = 1;

	for (i = 1; i <= n; i++)
	{
		total += flag*4.0/(2*i - 1);
		flag *= -1;
	}

	return total;
}

int main()
{
	int i;
	for (i	= 1; i <= 10000;i++)
	{
		printf("%d: %f\n",i,get_pi(i));
	}
	return 0;
}
```

3.3 请通过程序统计 1~100 中数字 9 的个数

```
#include <stdio.h>
int count_one(int num)
{
	int total = 0;
	while (num > 0)
	{
```

```
            if ((num % 10) == 9)
            {
                total++;
            }

            num = num / 10;
        }
        return total;
}

int count_9(int start,int end)
{
        int total = 0;
        int i = 0;

        for (i = start;i <= end; i++)
        {
            total = total + count_one(i);
        }

        return total;
}

int main()
{
        printf("%d\n",count_9( 1,100));
}
```

3.4 编程求 N 的阶乘(N!)

```
#include <stdio.h>

//方法一
int factorial1(int n)
{
        int total = 1;
        while (n > 0)
        {
            total *= n;
            n--;
        }
        return total;
}
```

```
//方法二
int factorial2(int n)
{
	int total = 1;
	int i = 1;
	for (i = 1; i <= n; i++)
	{
		total *= i;
	}

	return total;
}

//方法三
int factorial3(int n)
{
	if ((n == 0)||(n == 1))
	{
		return 1;
	}

	return n * factorial3( n - 1 );
}
int main()
{
	printf("%d\n",factorial3(5));
	return 0;
}
```

3.5 编程求 100 以内的素数，并打印

```
#include <stdio.h>
int is_prime(int num)
{
	if (num <= 1)
	{
		return 0;
	}

	int i;

	for (i = 2; i <= num/2 ; i++)
	{
```

```
            if (num % i == 0)
            {
                return 0;
            }
        }

        return 1;
}

int print_prime_num(int num)
{
        int i ;
        for (i = 1;i <= num; i++)
        {
            if (is_prime(i) == 1)
            {
                printf("%d ",i);
            }
        }
        printf("\n");
}

int main()
{
        print_prime_num(100);
}
```

3.6 求完数

一个数如果恰好等于它的因子之和,这个数被成为”完数”,例如:6=1+2+3. 请编程找出 1000 以内的完数

```
#include <stdio.h>
int is_perfect(int num)
{
        int total = 0;
        int i;

        for( i = 1; i < num; i++)
        {
            if ((num % i) == 0)
            {
                total += i;
            }
```

```
    }

    if (total == num)
    {
        return 1;
    }else
    {
        return 0;
    }
}

void print_perfect(int num)
{
    int total = 0;
    int i;

    printf("%d : ",num);
    for( i = 1; i < num; i++)
    {
        if ((num % i) == 0)
        {
            printf("%d ",i);
        }
    }
    printf("\n");
}

int count_perfect(int start,int end)
{
    int i;
    int total = 0;
    for (i = start; i <= end; i++)
    {
        if (is_perfect(i))
        {
            print_perfect(i);
            total++;
        }
    }

    return total;
}

int main()
```

```
{
    count_perfect(1,10000);
}
```

3.7 打印所有的水仙花数

水仙花数是指一个 3 位数 ，它的每个位上的数字的 3 次幂之和等于它本身（例如：$1^3 + 5^3 + 3^3 = 153$）

```
#include <stdio.h>

/*  判断num是否是水仙花数*/
int is_water_flower(int num)
{
    int total = 0;
    int tmp = num;
    int one = 0;

    if(num < 100    || num >= 1000 )
    {
        return 0;
    }

    while( tmp > 0 )
    {
        one = tmp % 10;
        total += one*one*one;
        tmp = tmp / 10;
    }

    return (total == num);
}

/*打印并统计所有的水仙花数  */
void print_count_water_flower()
{
    int total = 0;
    int i;

    for(i = 100; i < 1000; ++i)
    {
        if(is_water_flower(i))
        {
            printf("%d\n", i);
```

```
                ++total;
            }
        }

        printf("total = %d\n", total);
}

int main()
{
        print_count_water_flower();
}
```

4 函数与递归

4.1 可变参数

4.2 递归求 n 的阶乘（n!）

见 3.4 方法 3

4.3 求 Fibonacci （递归法）

Fibonacci数列：
0, 1, 1, 2, 3, 5, 8, 13, 21, …
用递归的方法求Fibonacci

```
#include <stdio.h>
int fabo(int n)
{
        if (n == 0)
        {
                return 0;
        }

        if (n == 1)
        {
                return 1;
        }

        return fabo(n - 1) + fabo(n - 2);
}
```

```
int main()
{
	int n;
	printf("Enter an integer:");
	scanf("%d",&n);
	printf("Fibonacci(%d) = %d\n",n,fabo(n));
}
```

4.4 最大公约数（递归与迭代）

两个整数的最大公约数（Greatest common divisor，GCD）是能够整除这两个整数的最大整数。请编写一个能够返回两个整数的最大公约数的gcd函数。

//使用迭代的方法

```
int	gcd (int a,int b)
{
	while (a != b)
	{
		if (a > b)
		{
			a = a - b;
		}
		else
		{
			b = b - a;
		}
    }

    return a;
}
```

//使用递归的方法

```
int gcd_r(int a,int b)
{
	if (a == b)
	{
		return a;
	}

	if (a > b)
	{
		gcd_r(a - b, b);
	}else
```

```
    {
        gcd_r(a,b - a);
    }
}
int main()
{
    printf("%d\n",gcd_r(30,15));
}
```

4.5 字符串反转（递归法）

用递归的方法将字符串反转：

```
void reverse_str(char* buf, int n)
{
    if(n < 2)
    {
        return;
    }

    char tmp = buf[0];
    buf[0] = buf[n-1];
    buf[n-1] = tmp;

    reverse_str(buf + 1, n - 2);
}

int main()
{
    char str[] = "helloworld";
    printf("%s\n",str);
    reverse_str(str,strlen(str));
    printf("%s\n",str);
}
```

4.6 二分法查找(递归法)

用递归的方法实现二分法查找：

```
#include <stdio.h>
int b_search(int *dest,int value,int low,int high)
{
    int mid;
    if (low > high)
```

```
    {
        return -1;
    }

    mid = (low + high) / 2;
    if (dest[mid] == value )
    {
        return mid;
    }

    if (value > dest[mid])
    {
        b_search(dest,value,mid + 1,high);
    }else
    {
        b_search(dest,value,low,mid - 1);
    }

}

int main()
{
    int dest[10] = {1,2,3,4,5,6,7,8,9,10};
    int value = 10;
    printf ("value: %d,postion: %d\n",value,b_search(dest,value,0,9));
    return 0;
}
```

4.7 汉诺塔


```
#include <stdio.h>
void tower(int from,int to,int aux,int n)
{
	if (n == 1)
	{
		printf("Move disk 1 from peg %c to peg %c\n",from,to);
		return ;
	}

	tower(from,aux,to,n - 1);
	printf("Move disk %d from peg %c to peg %c\n",n ,from,to);
	tower(aux,to,from,n - 1);
}

int main()
{
	tower('A','C','B',7);
}
```

5 数组

5.1 冒泡排序

```
#include<stdio.h>
void print_vec(int a[5],int n)
{
```

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

void bubble_sort(int *a,int num)
{
    int i = 0;
    int j = 0;
    int tmp;

    for (i = 0; i < num - 1; i++)
    {
        for (j = 0; j < num - 1 - i; j++)
        {
            if ( a[j] > a[j + 1])
            {
                tmp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = tmp;
            }
        }
    }

}

int main()
{
    int a[5] = {11,2,-31,14,59};
    print_vec(a,sizeof(a)/sizeof(a[0]));
    bubble_sort(a,sizeof(a)/sizeof(a[0]));
    print_vec(a,sizeof(a)/sizeof(a[0]));

    return 0;
}
```

5.2 二分法查找(迭代法)

```
#include <stdio.h>

int b_search(int *dest,int value,int len)
```

```
{
	int low = 0;
	int high = len - 1;
	int mid;

	while (low < high)
	{
		mid = (low + high)/2;

		if (dest[mid] == value)
		{
			return mid;
		}else if (value > dest[mid])
		{
			low = mid + 1;
		}else
		{
			high = mid - 1;
		}
	}

	return -1;
}

int main()
{
	int dest[10] = {1,2,3,4,5,6,7,8,9,10};
	int value = 9;
	printf ("value:%d,position:%d\n",value,b_search(dest,value,10));
	return 0;
}
```

5.3 memcpy

```
void *my_memcpy(void *dest,const void *src,int n)
{
	char *t_src = src;
	char *t_dest = dest;
	int i = 0;

	for ( i = 0; i < n; i++)
	{
		t_dest[i] = t_src[i];
	}
```

```
    return dest;
}
```

5.4 矩阵转置

```
#include <stdio.h>
void matrix(int vec[3][3])
{
    int i ;
    int j;
    int temp;
    for (i = 0; i < 3; i++)
    {
        for (j = i; j < 3; j++)
        {
            temp = vec[i][j];
            vec[i][j] = vec[j][i];
            vec[j][i] = temp;
        }
    }
}

void print_matrix(int (*vec)[3])
{
    int i;
    int j;
    for (i = 0; i < 3; i++)
    {
        for (j    = 0; j < 3; j++)
        {
            printf ("%d    ",vec[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

int main()
{
    int vec[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    print_matrix(vec);
    matrix(vec);
    print_matrix(vec);
```

```
}
```

6 指针

6.1 命令行参数

6.2 指向指针的指针

6.3 函数指针

7 字符串

7.1 strlen

```
int my_strlen(const char *str)
{
	int len = 0;
	while (*str != '\0')
	{
		len++;
		str++;
	}
	return len;
}
```

7.2 strcpy 与 strncpy

```
char *my_strcpy(char *dest,const char *src)
{
	if ((src == NULL) || (dest == NULL))
	{
		return NULL;
	}

	int i = 0;
	for ( i = 0; src[i] != '\0'; i++)
	{
		dest[i] = src[i];
```

```
    }
    dest[i] = '\0';

    return dest;
}

char *my_strncpy(char *dest,const char *src,int len)
{
    if ((src == NULL) || (dest == NULL))
    {
        return NULL;
    }

    int i = 0;
    for ( i = 0; (src[i] != '\0')&& (len - 1) > 0; i++)
    {
        dest[i] = src[i];
        len--;
    }
    dest[i] = '\0';

    return dest;
}

void my_strcpy2(char *dest,char *src)
{
    int i;
    while (*dest++ = *src++);
}

int main()
{
    char dest[30];
    char dest2[8];
    char src[] = "hellworld";
    printf("%s\n",my_strcpy(dest,src));
    printf("%s\n",my_strncpy(dest2,src,sizeof(dest2)));
}
```

7.3 atoi

```
#include <stdio.h>
int my_atoi(const char *str)
{
```

```
    int total = 0;
    int flag = 1;

    while (*str==' ')
    {
        str++;
    }

    if (*str == '-')
    {
        flag = -1;
        str++;
    }else if (*str == '+')
    {
        str++;
    }

    while((*str != '\0')&&(*str >= '0')&&(*str <= '9'))
    {
        total = total * 10 + *str - '0';
        str++;
    }

    return total*flag;
}


int main()
{
    char buff[30];
    printf("please enter an integer:");
    scanf("%s",buff);
    printf("%d\n",my_atoi(buff));
    return 0;
}
```

7.4 atof

```
#include <stdio.h>
double my_atof(const char *str)
{
    double total = 0;
    int flag = 1;
    int start_point = 0;
```

```
    int num_point = 0;
    int i = 0;

    while (*str==' ')
    {
        str++;
    }

    if (*str == '-')
    {
        flag = -1;
        str++;
    }else if (*str == '+')
    {
        str++;
    }

    while((*str != '\0')&&(*str >= '0')&&(*str <= '9')||(*str == '.'))
    {
        if (*str == '.')
        {
            str++;
            start_point = 1;
            continue;
        }
        if (start_point == 1)
        {
            num_point++;
        }
        total = total * 10 + *str - '0';
        str++;
    }

    for (i = num_point; i > 0; i--)
    {
        total = total / 10;
    }

    return total*flag;
}

int main()
{
    char buff[30];
```

```
    printf("Please enter a number:");
    scanf("%s",buff);
    printf("%f\n",my_atof(buff));
    return 0;
}
```

7.5 strcmp 与 strncmp

```
#include <stdio.h>
int my_strcmp(const char *s1,const char *s2)
{
    while (*s1 != '\0')
    {
        if (*s1 != *s2)
        {
            break;
        }
        s1++;
        s2++;
    }

    if (*s1 > *s2)
    {
        return 1;
    }else if (*s1 < *s2)
    {
        return -1;
    }else
    {
        return 0;
    }
}

int my_strncmp(const char *s1,const char *s2,int n)
{
    while ((*s1 != '\0')&&(n > 1))
    {
        if (*s1 != *s2)
        {
            break;
        }
        s1++;
        s2++;
        n--;
```

```
        }

        if (*s1 > *s2)
        {
                return 1;
        }else if (*s1 < *s2)
        {
                return -1;
        }else
        {
                return 0;
        }
}

int main()
{
        printf("%d\n",my_strcmp("hello","hello"));
        printf("%d\n",my_strncmp("hello","hello",5));
}
```

7.6 strstr

```
#include<string.h>
#include <stdio.h>
char *my_strstr(char *str,char *sub)
{
        int len = strlen(sub);
        while (*str != '\0')
        {
                if    (strncmp(str,sub,len) == 0)
                {
                        return str;
                }
                str++;
        }

        return NULL;
}

int main()
{
        printf("%s\n",my_strstr("hellookworld","ok"));
}
```

7.7 reverse

```
char *reverse(char *str)
{
	int len = strlen(str);
	int i;
	char temp;

	for (i = 0; i < len/2; i++)
	{
		temp = str[i];
		str[i] = str[len - i -1];
		str[len -i - 1] = temp;
	}
	return str;
}
```

7.8 strcat 与 strncat

```
#include <stdio.h>
#include <string.h>
char *my_strcat(char *dest,char *src)
{
	char *o_dest = dest;
	while (*dest != '\0')
	{
		dest++;
	}

	while(*src != '\0')
	{
		*dest++ = *src++;
	}

	*dest = '\0';

	return o_dest;
}

char*my_strncat(char *dest, const char *src, size_t n)
{
	size_t dest_len = strlen(dest);
	size_t i;
```

```
    for (i = 0 ; i < n && src[i] != '\0' ; i++)
    {
        dest[dest_len + i] = src[i];
    }

    dest[dest_len + i] = '\0';

    return dest;
}

int main()
{
    char a[20] = "helloworld";
    char *b = "hehe";
    printf("%s\n", strcat(a,b));
    printf("%s\n",strncat(a,b,2));
    return 0;
}
```

7.9 判断是字符串是否回文

```
#include <stdio.h>
#include <string.h>

//判断字符串是否为回文
int is_rev_str(const char *str)
{
    if (str == NULL)
    {
        return -1;
    }

    int len = strlen(str);
    int i = 0;
    for (i = 0; i < len / 2; i++)
    {
        if (str[i] != str[len - 1 - i])
        {
            return 0;          //不是回文
        }
    }
    return 1;                  //回文
}
```

```
int main()
{
	const char *str = "okko";
	printf("%d\n",is_rev_str(str));
}
```

8 位运算

8.1 以二进制打印无符号数

```
#include <stdio.h>
void print_bin(unsigned int x)
{
	unsigned int mask = 1 << 31;
	int count = 1;

	while (mask > 0)
	{
		if ((x & mask) > 0)
		{
			printf("1");
		}else
		{
			printf("0");
		}

		if (count == 4)
		{
			printf(" ");
			count = 0;
		}

		count ++;
		mask = mask >> 1;
	}
	printf("\n");
}

int main()
{
	print_bin(0xf);
}
```

8.2 统计一个无符号整数(unsigned int)的二进制表示中 1 的个数

```
#include <stdio.h>
int count_bit(unsigned int x)
{
	unsigned int mask = 1;
	int total = 0;

	while (x > 0)
	{
		total = total + (x & mask);
		x = x >> 1;
	}

	return total;
}

int count_bit2(unsigned int x)
{
	int total = 0;

	while (x > 0)
	{
		total = total + (x % 2);
		x = x / 2;
	}

	return total;
}

int main()
{
	printf("%d\n",count_bit(0xfe));
	printf("%d\n",count_bit2(0xfe));
}
```

8.3 循环右移

对一个 32 位无符号整数做循环右移，函数原型是unsigned int rotate_right(unsigned int x,int num);。所谓循环右移就是把低位移出去的部分再补到高位上去，例如

rotate_right(0xdeadbeef, 8)的值应该是 0xefdeadbe。

```
#include <stdio.h>
void print_bin(unsigned int x);

unsigned int rotate_right(unsigned int x,int num)
{
	unsigned int result = x;
	int i = 0;

	for (i = 0; i < num; i++)
	{
		result = ((result & 1) <<31) | (result >> 1);
	}

	 return result;
}

unsigned int rotate_right2(unsigned int x,int num)
{
	return (x << (32 - num)) | (x >> num);
}

int main()
{
	print_bin(0xdeadbeef);
	print_bin(rotate_right(0xdeadbeef,8));
	print_bin(rotate_right2(0xdeadbeef,8));
}
```

9 结构体、共用体、枚举与位字段

10 预编译与宏

11 综合训练

11.1 约瑟夫环(使用数组)

```
#include <assert.h>
```

```
#include <stdlib.h>
#include <stdio.h>
int josephus(int num)
{
	int left = num;
	int total = 0;
	int i = 0;

	int *p_people = (int *)malloc(sizeof(int));
	assert(p_people != NULL);

	for (i = 0; i < num; i++)
	{
		p_people[i] = i+1;
	}

	i = 0;
	while (left > 1)
	{
		if (p_people[i] > 0)
		{
			total++;
		}

		if (total == 3)
		{
			printf("%d ",p_people[i]);
			p_people[i] = 0;
			left--;
			total = 0;
		}

		i++;
		i = i % num;
	}
	printf("\n");

	for (i = 0; i < num; i++)
	{
		if (p_people[i] > 0)
		{
			return p_people[i];
		}
	}
```

```
    return 0;
}

int main(int argc,char *argv[])
{
    if (argc != 2)
    {
        printf("usage:./main people_num\n");
        return 2;
    }

    printf("%d \n",josephus(atoi(argv[1])));
    return 0;
}
```

11.2 约瑟夫环（使用环形链表）

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node
{
    int item;
    struct node *next;
}node;

node *create_link(int n)
{
    node *head = NULL;
    static node head_sentinel;
    head = &head_sentinel;
    head->item = 0;
    head->next = head;

    node *p;
    while (n > 0)
    {
        p = (node *)malloc(sizeof(node));
        if (p == NULL)
        {
            printf("create link:malloc failure\n");
            exit(1);
        }
```

```
            p->item = n;
            p->next = head->next;
            head->next = p;
            n--;
        }

        return head;
}

int jose(node *head,int n)
{
        int left = n;
        int total = 0;

        node *p = head->next;
        node *pre = head;
        int ret;

        while (left > 1 )
        {
            total++;
            if (total == 3)
            {
                printf("%d ",p->item);

                pre->next = p->next;
                free(p);

                p = pre->next;
                left--;
                total = 0;
            }

            pre = pre->next;
            p = p->next;
            if (p == head)
            {
                pre = head;
                p = head->next;
            }

        }

        ret = head->next->item;
```

```
    free(head->next);
    return ret;
}

int main()
{
    int num ;
    printf("please input the number of people:");
    scanf("%d",&num);
    node *head = create_link(num);
    printf("\nthe survival is: %d\n",jose(head,num));
}
```

11.3 银行卡号校验

当你输入信用卡号码的时候，有没有担心输错了而造成损失呢？其实可以不必这么担心，因为并不是一个随便的信用卡号码都是合法的，它必须通过Luhn算法来验证通过。

该校验的过程：

1、从卡号最后一位数字开始，逆向将奇数位(1、3、5 等等)相加。

2、从卡号最后一位数字开始，逆向将偶数位数字，先乘以 2（如果乘积为两位数，则将其减去 9），再求和。

3、将奇数位总和加上偶数位总和，结果应该可以被 10 整除。

例如，卡号是：5432123456788881

则奇数、偶数位分布：5432123456788881

奇数位和=35

偶数位乘以 2（有些要减去 9）的结果：1 6 2 6 1 5 7 7，求和=35。

最后 35+35=70 可以被 10 整除，认定校验通过。

请编写一个程序，从键盘输入卡号，然后判断是否校验通过。通过显示：“成功”，否则显示“失败”。

比如，用户输入：356827027232780

程序输出：成功

```
#include <stdio.h>
#include <string.h>

int is_right(char *card_no)
{
	int len = strlen(card_no);
	int total = 0;
	int tmp;
	int i;

	for ( i = len - 1; i >= 0; i = i - 2)
	{
		total += card_no[i] - '0';
	}

	for (i = len - 2; i >= 0;i =i - 2)
	{
		tmp = (card_no[i] - '0')*2;
		if (tmp > 9)
		{
			tmp = tmp - 9;
		}
		total += tmp;
	}

	if (total % 10 == 0)
	{
		return 1;
	}else
	{
		return 0;
	}

}

int main()
{
	char card_no[30];

	printf("please input your card number.\n");
	scanf("%s",card_no);
	if (is_right(card_no) == 1)
	{
		printf("success\n");
```

```
    }else
    {
        printf("fail\n");
    }
}
```

11.4 单词逆序

编写一个C函数,将”I am from shanghai ”倒置为”shanghai from am I”,及将句子中的单词位置倒置,而不改变单词内部结构.

```
#include <stdio.h>
char *reverse(char *str)
{
    int len = strlen(str);
    int i;
    char temp;

    for (i = 0; i < len/2; i++)
    {
        temp = str[i];
        str[i] = str[len - i -1];
        str[len -i - 1] = temp;
    }
    return str;
}

int word_len(char *str)
{
    int len = 0;
    while ((*str != '\0') && *str != ' ')
    {
```

```
            len++;
            str++;
        }

        return len;
}

char *reverse_word(char *str)
{
        int len = word_len(str);
        int i;
        char temp;

        for (i = 0; i < len/2; i++)
        {
            temp = str[i];
            str[i] = str[len - i -1];
            str[len -i - 1] = temp;
        }

        return str + len;
}

char *reverse_only(char *src)
{
        char *src_temp = src;
        reverse(src);
        while (*src != '\0')
        {
```

```
            while(*src == ' ')
            {
                src++;
            }
            src = reverse_word(src);
        }

        return src_temp;
}

int main(int argc,char *argv[])
{
        char src[] = "I am from Shanghai";
        printf("%s\n",reverse_only(src));
}
```

11.5 进制转换

请编写一个C函数,该函数可以实现将一个整数转为任意进制的字符串输出

```
#include <stdio.h>
#include <string.h>
#include <assert.h>
char *reverse(char *str)
{
        int len = strlen(str);
        int i;
        char temp;

        for (i = 0; i < len/2; i++)
```

```
    {
        temp = str[i];
        str[i] = str[len - i -1];
        str[len -i - 1] = temp;
    }
    return str;
}

char *trans(char *dest,unsigned int input,int scale)
{
    int i = 0 ;
    int mod;
    assert(scale > 1);
    while (input > 0)
    {
        mod = input % scale;
        if (mod < 10)
        {
            dest[i] = mod + '0';
        }else
        {
            dest[i] = mod - 10 + 'a';
        }
        i++;
        input = input / scale;
    }
    dest[i] = '\0';
    reverse(dest);
    return dest;
```

```
}


int main(int argc,char *argv[])
{
	char a[32];
	printf("%s\n",trans(a,255,16));
	printf("%s\n",trans(a,255,2));
}
```

第二部分 数据结构

1 线性表

1.1 单向链表

不带头结点单向链表的从头结点插入、顺序插入、删除、遍历、查找、逆序和销毁

```
//link.h
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
	int item;
	struct node *next;
}node;
node * mk_node(int item);
void free_node(node *p);
void insert(node *p);
void insert_sort_l2p(node *p);
void traverse();
node *find(int item);
int delete_node(node *p);
void destroy();
void reverse();
//link.c
```

```
#include "link.h"

node * head = NULL;

node * mk_node(int item)
{
	node *p = (node *)malloc(sizeof(node));
	p->item = item;
	return p;
}

void free_node(node *p)
{
	free(p);
}

void insert(node *p)
{
	p->next = head;
	head    = p;
}

void insert_sort_l2b(node *p)
{
	if (head == NULL)
	{
		p->next =head;
		head = p;
		return ;
	}
	if (p->item < head->item)
	{
		p->next = head;
		head = p;
		return ;
	}
	node *pre = head;
	while (pre->next != NULL)
	{
		if ((p->item > pre->item)&&(p->item < pre->next->item))
		{
			p->next = pre->next;
			pre->next = p;
			return ;
```

```
            }
            pre = pre->next;
        }

        p->next = pre->next;
        pre->next = p;
        return ;
}

void traverse()
{
        node *p = head;
        while (p != NULL)
        {
            printf("%d    ",p->item);
            p = p->next;
        }
        printf("\n-------------------\n");
}

node *find(int item)
{
        node *p = head;
        while (p != NULL)
        {
            if (p->item == item)
            {
                return p;
            }
            p = p->next;
        }
        return NULL;
}

int delete_node(node *p)
{
        if (head == p)
        {
            head = head->next;
            return 1;
        }

        node *pre = head;
        while (pre->next != NULL)
```

```
        {
            if (pre->next == p)
            {
                pre->next = p->next;
                return 1;
            }
            pre = pre->next;
        }
        return 0;
}

void destroy()
{
        node *p;
        while (head != NULL)
        {
            p = head;
            head = head->next;
            free_node(p);
        }
}

void reverse()
{
        node *nhead = NULL;

        node *p;
        while (head != NULL)
        {
            p = head;
            head = head->next;

            p->next = nhead;
            nhead    = p;
        }

        head = nhead;
}

//main.c
#include <stdio.h>
#include "link.h"
int main()
{
```

```
	node *p = mk_node(1);
	insert_sort_l2b(p);
	traverse();

	p = mk_node(-1);
	insert_sort_l2b(p);
	traverse();

	p = mk_node(0);
	insert_sort_l2b(p);
	traverse();

	p = mk_node(-3);
	insert_sort_l2b(p);
	traverse();

	p = mk_node(7);
	insert_sort_l2b(p);
	traverse();

	p = find(1);
	if (p != NULL)
	{
		delete_node(p);
		free_node(p);
	}else
	{
		printf("can't find %d\n",1);
	}

	traverse();
	reverse();
	traverse();
}
```

1.2 双向链表

带头结点的双向链表

```
//dlink.h
typedef struct node
{
	int item;
	struct node * pre;
	 struct node * next;
```

```
}node;

node * make_node(int item);
void free_node(node * p);
node * search(int item);
void insert_node(node * p);
void delete_node(node * p);
void traverse();
void destroy(void);
void enqueue(node * p);
node * dequeue(void);

//dlink.c
#include <stdlib.h>
#include <stdio.h>
#include "dlink.h"

struct node tail_sentinel;
struct node head_sentinel = {0, NULL, &tail_sentinel};
struct node tail_sentinel = {0, &head_sentinel, NULL};

static node * head = &head_sentinel;
static node * tail = &tail_sentinel;

node * make_node(int item)
{
	node* p = malloc(sizeof *p);
	 if (p == NULL)
	 {
		return NULL;
	 }

	p->item = item;
	p->pre = p->next = NULL;
	return p;
}

void free_node(node * p)
{
	free(p);
}

node * search(int item)
{
```

```
    node * p;
    for (p = head->next; p != tail; p = p->next)
    {
        if (p->item == item)
        {
            return p;
        }
    }

    return NULL;
}

void insert_node(node *p)
{
    p->next = head->next;
    head->next->pre = p;
    head->next = p;
    p->pre = head;
}

void delete_node(node *p)
{
    p->pre->next = p->next;
    p->next->pre = p->pre;
}

void traverse()
{
    node * p;
    for (p = head->next; p != tail; p = p->next)
    {
        printf("%d    ",p->item);
    }
     printf("\n---------------------------------\n");
}

void destroy(void)
{
    node *q;
     node *p = head->next;
    head->next = tail;
    tail->pre = head;

    while (p != tail)
```

```
    {
        q = p;
        p = p->next;
        free_node(q);
    }
}

void enqueue(node * p)
{
    insert_node(p);
}

node * dequeue(void)
{
    if (tail->pre == head)
    {
        return NULL;
    }
    else
    {
        node *p = tail->pre;
        delete_node(p);
        return p;
    }
}

//main.c
#include <stdio.h>
#include "dlink.h"

int main(void)
{
    node * p = make_node(10);
    insert_node(p);
    p = make_node(5);
    insert_node(p);
    p = make_node(90);
    insert_node(p);

     traverse();

     p = search(5);
    delete_node(p);
    free_node(p);
```

```
    traverse();
    destroy();

    p = make_node(2);
    enqueue(p);
    p = make_node(5);
    enqueue(p);
    p = make_node(10);
    enqueue(p);
    while (p = dequeue())
      {
          printf("%d ",p->item);
        free_node(p);
    }
     printf("\n");

    return 0;
}
```

1.3 循环链表

带头结点的单向循环链表

```
//looplink.h
#include <stdio.h>
#include <stdlib.h>
typedef struct node
{
     int item;
     struct node *next;
}node;

void init_link();
node *mk_node(int item);
void free_node(node *p);
void insert_node(node *p);
node *find_item(int item);
int delete_node(node *p);
void traverse();
void destroy();

//looplink.c
#include "looplink.h"
static node head_sentinel={0,NULL};
```

```
node *head = NULL;

void init_link()
{
	head = &head_sentinel;
	head_sentinel.next = head;
}

node *mk_node(int item)
{
	node *p = (node *)malloc(sizeof(node));
	if (p == NULL)
	{
		printf("mk_node:malloc failure\n");
		exit(1);
	}
	p->item = item;
	p->next = NULL;
	return p;
}

void free_node(node *p)
{
	free(p);
}

void insert_node(node *p)
{
	p->next = head->next;
	head->next = p;
}

node *find_item(int item)
{
	node* p = head->next;
	while (p != head)
	{
		if (p->item == item)
		{
			return p;
		}
		p = p->next;
	}
```

```
    return NULL;
}

int delete_node(node *p)
{
    node *pre = head;
    while (pre->next !=    head)
    {
        if (pre->next == p)
        {
            pre->next = p->next;
            return 1;
        }
        pre = pre->next;
    }
    return 0;
}

void traverse()
{
    node *p = head->next;
    while (p != head)
    {
        printf("%d ",p->item);
        p = p->next;
    }
    printf("\n");
}

void destroy()
{
    node * p = head->next;
    while (p != head)
    {
        printf("%d ",p->item);
        delete_node(p);
        free_node(p);
        p = p->next;
    }
}

//main.c
#include "looplink.h"
```

```
int main()
{
	init_link();

	node *p = mk_node(1);
	insert_node(p);
	traverse();

	p = mk_node(2);
	insert_node(p);
	traverse();

	p = mk_node(3);
	insert_node(p);
	traverse();

	p = mk_node(4);
	insert_node(p);
	traverse();

	p = find_item(4);
	delete_node(p);
	free_node(p);
	traverse();

	p = find_item(3);
	delete_node(p);
	free_node(p);
	traverse();

	p = find_item(2);
	delete_node(p);
	free_node(p);
	traverse();

	p = find_item(1);
	delete_node(p);
	free_node(p);
	traverse();

	destroy();
}
```

2 栈

2.1 用链表实现栈

```
//stack.h
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
	int item;
	struct node *next;
}node;
node * mk_node(int item);
void free_node(node *p);
void insert(node *p);
void traverse();
node *find(int item);
int delete_node(node *p);
void destroy();
void push(int);
int pop();
int is_empty();

//stack.c
#include "stack.h"

node * head = NULL;

node * mk_node(int item)
{
	node *p = (node *)malloc(sizeof(node));
	p->item = item;
	return p;
}

void free_node(node *p)
{
	free(p);
}

void insert(node *p)
{
```

```
    p->next = head;
    head    = p;
}

void traverse()
{
    node *p = head;
    while (p != NULL)
    {
        printf("%d    ",p->item);
        p = p->next;
    }
    printf("\n-------------------\n");
}

node *find(int item)
{
    node *p = head;
    while (p != NULL)
    {
        if (p->item == item)
        {
            return p;
        }
        p = p->next;
    }
    return NULL;
}

int delete_node(node *p)
{
    if (head == p)
    {
        head = head->next;
        return 1;
    }

    node *pre = head;
    while (pre->next != NULL)
    {
        if (pre->next == p)
        {
            pre->next = p->next;
            return 1;
```

```
            }
            pre = pre->next;
        }
        return 0;
}

void destroy()
{
        node *p;
        while (head != NULL)
        {
            p = head;
            head = head->next;
            free_node(p);
        }
}

void push(int i)
{
        node *p = mk_node(i);
        insert(p);
}

int pop()
{
        node * p = head;
        head = head->next;
        int item = p->item;
        free_node(p);

        return item;
}

int is_empty()
{
        if (head == NULL)
        {
            return 1;
        }else
        {
            return 0;
        }
}
```

```
//main.c
#include <stdio.h>
#include "stack.h"
int main()
{
	push(1);
	push(2);
	push(3);
	while (!is_empty())
	{
		printf("%d\n",pop());
	}
}
```

2.2 用数组实现栈

```
#include <stdio.h>
#define STACK_SIZE 512

char stack[STACK_SIZE];

int top = 0;

void push(char c)
{
	stack[top] = c;
	top++;
}

char pop(void)
{
	return stack[--top];
}

int is_empty(void)
{
	return top == 0;
}

int is_full(void)
{
	return top == STACK_SIZE;
}
```

```
int main(void)
{
	push('a');
	push('b');
	push('c');

	while(!is_empty())
	{
		printf("%c ",pop());
	}
	printf("\n");

	return 0;
}
```

2.3 用栈走迷宫（深度优先）

```
#include <stdio.h>
#define STACK_SIZE 512
#define MAX_ROW 5
#define MAX_COL 5
#define PASSED  2

typedef struct point
{
	int row;
	int col;
}point;

point stack[STACK_SIZE];

int top = 0;

void push(struct point p)
{
	stack[top++] = p;
}

point pop(void)
{
	return stack[--top];
}

int is_empty(void)
```

```
{
	return top == 0;
}

int is_full(void)
{
	return top == STACK_SIZE;
}

int maze[MAX_ROW][MAX_COL] =
{
	0, 1, 0, 0, 0,
	0, 1, 0, 1, 0,
	0, 0, 0, 1, 0,
	0, 1, 1, 1, 0,
	0, 0, 0, 1, 0,
};

void print_maze(void)
{
	int i, j;
	for (i = 0; i < MAX_ROW; i++)
	{
		for (j = 0; j < MAX_COL; j++)
		{
			printf("%d ", maze[i][j]);
		}
		putchar('\n');
	}
	printf("------------------------\n");
}

point predecessor[MAX_ROW][MAX_COL] =
{
	{{-1,-1}, {-1,-1}, {-1,-1}, {-1,-1}, {-1,-1}},
	{{-1,-1}, {-1,-1}, {-1,-1}, {-1,-1}, {-1,-1}},
	{{-1,-1}, {-1,-1}, {-1,-1}, {-1,-1}, {-1,-1}},
	{{-1,-1}, {-1,-1}, {-1,-1}, {-1,-1}, {-1,-1}},
	{{-1,-1}, {-1,-1}, {-1,-1}, {-1,-1}, {-1,-1}},
};

void visit(int row, int col, point pre)
{
	point visit_point = { row, col };
```

```
    maze[row][col] = PASSED;
    predecessor[row][col] = pre;
    push(visit_point);
}

int main(void)
{
    struct point p = { 0, 0 };

    maze[p.row][p.col] = PASSED;
    push(p);

    while (!is_empty())
    {
        p = pop();

        //goal
        if (p.row == MAX_ROW - 1 && p.col == MAX_COL - 1)
        {
            break;
        }
        //right
        if (p.col + 1 < MAX_COL && maze[p.row][p.col + 1] == 0)
        {
            visit(p.row, p.col + 1, p);
        }

        //down
        if (p.row + 1 < MAX_ROW && maze[p.row + 1][p.col] == 0)
        {
            visit(p.row + 1, p.col, p);
        }

        //left
        if (p.col-1 >= 0 && maze[p.row][p.col - 1] == 0)
        {
            visit(p.row, p.col - 1, p);
        }

        //up
        if (p.row - 1 >= 0 && maze[p.row - 1][p.col] == 0)
        {
            visit(p.row - 1, p.col, p);
        }
```

```
            print_maze();
        }

        if (p.row == MAX_ROW - 1 && p.col == MAX_COL - 1)
        {
            printf("(%d, %d)\n", p.row, p.col);
            while (predecessor[p.row][p.col].row != -1)
            {
                p = predecessor[p.row][p.col];
                printf("(%d, %d)\n", p.row, p.col);
            }
        } else
        {
            printf("No path!\n");
        }

        return 0;
}
```

3 队列

3.1 用链表实现队列

```
//queue.h
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int item;
    struct node *next;
}node;
node * mk_node(int item);
void free_node(node *p);
void insert(node *p);
void traverse();
node *find(int item);
int delete_node(node *p);
void destroy();
int is_empty();
void enqueue(int);
int dequeue();
```

```
//queue.c
#include "queue.h"

node * head = NULL;

node * mk_node(int item)
{
	node *p = (node *)malloc(sizeof(node));
	p->item = item;
	return p;
}

void free_node(node *p)
{
	free(p);
}

void insert(node *p)
{
	p->next = head;
	head    = p;
}

void traverse()
{
	node *p = head;
	while (p != NULL)
	{
		printf("%d    ",p->item);
		p = p->next;
	}
	printf("\n-------------------\n");
}

node *find(int item)
{
	node *p = head;
	while (p != NULL)
	{
		if (p->item == item)
		{
			return p;
		}
```

```
            p = p->next;
        }
        return NULL;
}

int delete_node(node *p)
{
        if (head == p)
        {
            head = head->next;
            return 1;
        }

        node *pre = head;
        while (pre->next != NULL)
        {
            if (pre->next == p)
            {
                pre->next = p->next;
                return 1;
            }
            pre = pre->next;
        }
        return 0;
}

void destroy()
{
        node *p;
        while (head != NULL)
        {
            p = head;
            head = head->next;
            free_node(p);
        }
}

int is_empty()
{
        if (head == NULL)
        {
            return 1;
        }else
        {
```

```
            return 0;
        }
}

void enqueue(int item)
{
        node *p = mk_node(item);
        if (head == NULL)
        {
            p->next = head;
            head = p;
            return ;
        }

        node *tail = head;
        while(tail->next != NULL)
        {
            tail = tail->next;
        }

        p->next = tail->next;
        tail->next = p;
}

int dequeue()
{
        node * p = head;
        head = head->next;
        int item = p->item;
        free_node(p);

        return item;
}

//main.c
#include <stdio.h>
#include "queue.h"
int main()
{
        enqueue(1);
        enqueue(2);
        enqueue(3);
        while (!is_empty())
        {
```

```
            printf("%d\n",dequeue());
        }
}
```

3.2 用队列走迷宫（广度优先）

```
#include <stdio.h>
#define QUEUE_SIZE 512
#define MAX_ROW 5
#define MAX_COL 5
#define PASSED 2

typedef struct point
{
    int row;
    int col;
    int predecessor;
}point;

point queue[QUEUE_SIZE];

int head = 0;
int tail = 0;

void enqueue(point p)
{
    queue[tail++] = p;
}

point dequeue(void)
{
    return queue[head++];
}

int is_empty(void)
{
    return head == tail;
}

int maze[MAX_ROW][MAX_COL] =
{
    0, 1, 0, 0, 0,
    0, 1, 0, 1, 0,
    0, 0, 0, 0, 0,
```

```
    0, 1, 1, 1, 0,
    0, 0, 0, 1, 0,
};

void print_maze(void)
{
    int i, j;
    for (i = 0; i < MAX_ROW; i++)
    {
        for (j = 0; j < MAX_COL; j++)
        {
            printf("%d ", maze[i][j]);
        }
        putchar('\n');
    }
    printf("-----------------\n");
}

void visit(int row, int col)
{
    point visit_point = { row, col, head - 1 };
    maze[row][col] = PASSED;
    enqueue(visit_point);
}

int main(void)
{
    struct point p = { 0, 0, -1 };

    maze[p.row][p.col] = PASSED;
    enqueue(p);

    while (!is_empty())
    {
        p = dequeue();

        //goal
        if (p.row == MAX_ROW - 1 && p.col == MAX_COL - 1)
        {
            break;
        }

        //right
        if (p.col + 1 < MAX_COL && maze[p.row][p.col + 1] == 0)
```

```
        {
            visit(p.row, p.col+1);
        }

        //down
        if (p.row + 1 < MAX_ROW && maze[p.row + 1][p.col] == 0)
        {
            visit(p.row + 1, p.col);
        }

        //left
        if (p.col - 1 >= 0 && maze[p.row][p.col - 1] == 0)
        {
            visit(p.row, p.col - 1);
        }

        //up
        if (p.row - 1 >= 0 && maze[p.row - 1][p.col] == 0)
        {
            visit(p.row - 1, p.col);
        }

        print_maze();
    }

    if (p.row == MAX_ROW - 1 && p.col == MAX_COL - 1)
    {
        printf("(%d, %d)\n", p.row, p.col);
        while (p.predecessor != -1)
        {
            p = queue[p.predecessor];
            printf("(%d, %d)\n", p.row, p.col);
        }
    } else
    {
        printf("No path!\n");
    }

    return 0;
}
```

4 二叉树

实现二叉树顺序插入、中序遍历、前序遍历、后序遍历、查找

```
#include<stdlib.h>
#include <stdio.h>
typedef struct node
{
	int item;
	struct node *left;
	struct node * right;
}node;

node *root = NULL;
void insert_node(int item)
{
	node *pre;
	node *parent;
	node *p = (node *)malloc(sizeof(node));
	if (p == NULL)
	{
		printf("insert_node:malloc failure\n");
		exit(1);
	}

	p->item = item;
	p->left = NULL;
	p->right = NULL;

	parent = pre = root;
	if (root == NULL)
	{
		root = p;
		return ;
	}

	while (pre != NULL)
	{
		parent = pre;
		if (item > pre->item)
		{
			pre = pre->right;
		}else if(item < pre->item)
		{
			pre = pre->left;
		}else
		{
```

```
                free(p);
                return;
            }
        }

        if (p->item > parent->item)
        {
            parent->right = p;
        }else
        {
            parent->left = p;
        }
}

void traverse_m(node *p)
{
        if (p == NULL)
        {
            return ;
        }
        traverse_m(p->left);
        printf("%d ",p->item);
        traverse_m(p->right);
}

void traverse_b(node *p)
{
        if (p == NULL)
        {
            return ;
        }
        printf("%d ",p->item);
        traverse_b(p->left);
        traverse_b(p->right);
}

void traverse_a(node *p)
{
        if (p == NULL)
        {
            return ;
        }
        traverse_a(p->left);
        traverse_a(p->right);
```

```
	printf("%d ",p->item);
}

node *search_node(node *current,int item)
{
	if (item < current->item)
	{
		if (current->left == NULL)
		{
			return NULL;
		}
		return search_node(current->left,item);
	}else if(item > current->item)
	{
		if (current->right == NULL)
		{
			return NULL;
		}
		return search_node(current->right,item);
	}

	return current;
}

int main()
{
	insert_node(8);
	insert_node(4);
	insert_node(9);
	insert_node(13);
	insert_node(11);
	printf("中序遍历\n");
	traverse_m(root);
	printf("\n");
	printf("后序遍历\n");
	traverse_a(root);
	printf("\n");
	printf("前序遍历\n");
	traverse_b(root);
	printf("\n");
	int item ;
	printf("please input you item:\n");
	scanf("%d",&item);
```

```
    node *p = search_node(root,item);
    if (p != NULL)
    {
        printf("node:%p: item %d\n",p,p->item);
    }else
    {
        printf("can't find %d in this tree\n",item);
    }
}
```

5 查找

5.1 顺序查找

5.2 二分法查找

6 排序

6.1 冒泡排序

见C语言 5.1 冒泡排序例子

6.2 插入排序

```
#include<stdio.h>
void print_vec(int a[5],int n)
{
    int i = 0;
    for (i    = 0; i < n; i++)
    {
        printf("%d ",a[i]);
    }
    printf("\n");
}

void insert_sort(int *a,int num)
{
    int i = 0;
    int j = 0;
```

```
    int tmp;

    for (i = 1; i < num ; i++)
    {
        tmp = a[i];
        for (j = i - 1; (j >= 0) && (tmp < a[j]); j-- )
        {
            a[j + 1] = a[j];
        }

        a[j + 1] = tmp;
          print_vec(a,5);
    }

}

int main()
{
    int a[5] = {9,8,7,6,5};
    print_vec(a,sizeof(a)/sizeof(a[0]));
    insert_sort(a,sizeof(a)/sizeof(a[0]));
    print_vec(a,sizeof(a)/sizeof(a[0]));

    return 0;
}
```

6.3 选择排序

```
#include<stdio.h>
void print_vec(int a[5],int n)
{
    int i = 0;
    for (i    = 0; i < n; i++)
    {
        printf("%d ",a[i]);
    }
    printf("\n");
}

//选择排序
void select_sort(int *v,int len)
{
    int i;
```

```
    int j;
    int v_min;
    int p_min;

    for (i = 0; i < len; i++)
    {
        v_min = v[i];
        p_min = i;

        for (j = i; j < len; j++)
        {
            if (v[j] < v_min)
            {
                v_min = v[j];
                p_min = j;
            }
        }

        v[p_min] = v[i];
        v[i] = v_min;
    }
}

int main()
{
    int a[5] = {9,8,7,6,5};
    print_vec(a,sizeof(a)/sizeof(a[0]));
    select_sort(a,sizeof(a)/sizeof(a[0]));
    print_vec(a,sizeof(a)/sizeof(a[0]));

    return 0;
}
```

6.4 shell 排序

```
#include<stdio.h>
void print_vec(int a[5],int n)
{
    int i = 0;
    for (i    = 0; i < n; i++)
    {
        printf("%d ",a[i]);
    }
```

```
    printf("\n");
}

void shell_sort(int *a,int len)
{
    int i = 0;
    int j = 0;
    int h = 0;
    int tmp;
    for (h = len / 2; h > 0; h = h /2)
    {
        for (i = h; i < len ; i++)
        {
            tmp = a[i];
            for (j = i - h; (j >= 0) && (tmp < a[j]); j -= h )
            {
                a[j + h] = a[j];
            }
            a[j + h] = tmp;
        }
    }
}

int main()
{
    int a[5] = {9,8,7,6,5};
    print_vec(a,sizeof(a)/sizeof(a[0]));
    shell_sort(a,sizeof(a)/sizeof(a[0]));
    print_vec(a,sizeof(a)/sizeof(a[0]));

    return 0;
}
```

6.5 快速排序

```
#include<stdio.h>
void print_vec(int a[5],int n)
{
    int i = 0;
    for (i    = 0; i < n; i++)
    {
        printf("%d ",a[i]);
    }
```

```
	printf("\n");
}

void swap(int *v,int i,int j)
{
	int tmp;
	tmp = v[i];
	v[i] = v[j];
	v[j] = tmp;
}

//快速排序
void q_sort(int *v,int left,int right)
{
	int i = 0;
	int last;

	if (left >= right)
	{
		return ;
	}
	swap(v,left,(left + right) / 2);
	last = left;
	for (i = left + 1; i <= right;i++)
	{
		if (v[i] < v[left])
		{
			swap(v,++last,i);
		}
	}
	swap(v,left,last);
	q_sort(v,left,last - 1);
	q_sort(v,last + 1,right);

}

int main()
{
	int a[5] = {9,8,7,6,5};
	print_vec(a,sizeof(a)/sizeof(a[0]));
	q_sort(a,0,sizeof(a)/sizeof(a[0]) -1);
	print_vec(a,sizeof(a)/sizeof(a[0]));

	return 0;
```

```
}
```