一、指出程序段中的错误：分析错误的原因，并进行修改。(每个5分，共10分)

1、函数swap将两个字符串(字符数组作实参，长度不超过100)的内容进行交换。

Void swap(char\*pa，char\*pb)

{

char\*temp;

temp=pa;

pa=pb;

pb=temp;

}

//

指针temp没有指向一个确定的内存单元，改正：   
Void change(char \*p1,char \*p2)  
     {  
        Char \*temp;  
                Char str[80];    //编译器会给str开辟内存的  
                Temp=str;  
        Temp=p1;  
         P1=p2;  
                 P2=temp;  
     }

千万要注意不可写成这个样子：

Void change(char \*p1,char \*p2)

{

Char str[80]; //编译器会给str开辟内存的

Str=p1;

P1=p2;

P2=str;

}

以上的一些写于成都，为什么不写了呢？我想一些高人们可能也看出我的水平来了，我的第一个题就做错了。其实char \* temp;这句没有任何问题，他在考察的是值传递。指针传过去也是值传递。所谓的地址传递其实是将实参中的指针所指向的内容由形参（即被调用函数）来改变。还是谭浩强的那个经典的例子，这里说一句：谭浩强的哪本书真的很好啊，很好，王道！下面将这个例子描述如下：  
#include <stdio.h>  
void main()  
{  
void swap(int \*p1,int \*p2);  
int a,b;  
a=3;  
b=5;  
printf("交换前a=%d,b=%d\n",a,b);  
swap(&a,&b);  
printf("交换后a=%d,b=%d\n",a,b);  
}  
void swap(int \*p1,int \*p2)  
{  
        int temp;  
        temp=\*p1;  
        \*p1=\*p2;  
        \*p2=temp;  
}  
这个程序都见过吧，这个里面到底发生了什么，自己去谭浩强课本中看吧。。这个电大的第一道题是让你写一个程序将两个字符串交换，这个就不难了哈，我是耍小聪明了，唉~~  
第一个题写了很多，下面的就少写点了哈。。。

//

方法1: void swap(char \*\*pa,char \*\*pb)

{

char \*temp;

temp=\*pa;

\*pa=\*pb;

\*pb=temp;

}

方法2: void swap(char \*pa,char \*pb)

{

char temp[100];//题目说明最大为100字节

strcpy(temp,pa);

strcpy(pa,pb);

strcpy(pb,temp);

}

//

答：1.函数类型大小写错误。

2. 交换形参指针地址不能修改实参的值。

修改：

void swap(char \*pa, char \*pb){

char temp[MAX];

int i;

for(i=0;pa[i]!='\0';++i)

temp[i]=pa[i];

temp[i]=pa[i];

for(i=0;pb[i]!='\0';++i)

pa[i]=pb[i];

pa[i]=pb[i];

for(i=0;temp[i]!='\0';++i)

pb[i]=temp[i];

pb[i]=temp[i];

}

或使用库函数：

void swap(char \*pa, char \*pb){

char temp[MAX];

int i;

strcpy(temp,pa);

strcpy(pa,pb);

strcpy(pb,temp);

}

2、程序片段为：

charpa[]=“ABCDE”;

char\*pb=“EFG”;

pb[1]=‘A’; //pb指向的内容是常量，不能修改

pb=pa;

strcpy(pa,“ABCDEFGXYZ”); //pa指向内存空间不够

pb=“D”;

//

.其实第二个我在第一题中有所描述，就是不能给数组名赋值的。  
如：char str[50];  
    str="I love";  
是错误的，因为str是个常量。。

//

答：1. pb[1]='A'尝试修改常量，出错

2. strcpy函数尝试将一个长度超过pa的字符串复制进pa

修改代码。【我都没看出来这代码要干什么，怎么改？】

二、简答题(共30分)

1、设arr为整型数组，num和item为整型变量，N=数组元素个数-1。需要查找item是否在数组中，如果程序片段为

for(num=N;arr[num]!=item;num–);

printf(“%d”,num);

可能导致的异常结果是什么?为什么?(8分)

//

这里有越界的危险哈，有可能在数组中根本就没有给定的值，则i会一直在减1。

//

结果可能是输出一个负值，因为item是个变量，分配了存储空间， arr[-2]也是可以输出的，因为arr只是个地址，经过测试会找到item，printf他的地址

//

答：可能的异常结果为出现一个随机负数。

由于代码未检查数组是否越界，若查找的item未在数组中，则会导致程序超出数组范围继续寻找，由于item为变量，也存在于程序的内存中，最终若在数组界外查到item，则输出一个负数。【若item为全局变量，由于存储位置不同，可导致内存查找越界，而异常退出。感谢Seven同学的指导啊！】

2、设有递归函数：

Intvalue(intn){

intx;

if(n==0)

return0;

else

{

scanf(“%d”,&x);

return(value(n-1)+x);

}

}

如该函数被调用时，参数n值为4，输入的x的值依次为11，22，33，44，函数调用结束时返回值是多少?并用图描述函数递归执行过程。(10分)

//

还是谭浩强那本书，第172页把这页的图类似的画上就行了。

结果是110， 图参考谭浩强的递归那节

答：返回值为110



3、数组作为函数参数有三种形式：1)实参是数组元素：2)形参是指针，实参是函数：3)函数的形参和实参都是数组分别是采用什么参数传递方式?(5分)

1)实参是数组元素 传值

2)形参是指针 传地址(指针)

3)形参和实参都是数组 传地址

//

答：1）传递数组元素的值；2）传递数组的起始地址指针；3）传递数组的起始地址

4、采用高度抽象概念有利于程序设计，C语言中循环语句

do

s

while(B);

对应的显示控制结构是什么?请使用伪代码形式(通过条件转移指令)表达。(7分)

Answer: label: s;

If(B) goto label;

三、程序设计(共45分，每题的算法50%，语法30%，完整20%)

1、编写完整程序：利用2个函数对输入的两个分数进行加、减、乘、除四则运算和输出用分数表示的结果。(注：输入格式为：%ld/%ld%c%ld/%ld，输出格式为%ld/%ld)，例如：输入1/4+1/3，输出：7/12(10分)

//

(这道题要考虑周全得鲁很多码，见谅意思意思就行)

#include<stdio.h>

long gcd(long m,long n)

{

while(n)

{

long t=m%n;

m=n;

n=t;

}

return m;

}

void count(char flag,long up1,long up2,long down1,long down2)

{

long t;

switch(flag)

{

case '+':

up1=up1\*down2+up2\*down1;

down1=down1\*down2;

break;

case '-':

up1=up1\*down2-up2\*down1;

down1=down1\*down2;

break;

case '\*':

up1=up1\*up2;

down1=down1\*down2;

break;

case '/':

up1=up1\*down2;

down2=down1\*up2;

break;

}

t=gcd(up1,down1);

if(up1==down1)

printf("1\n");

else

printf("%d/%d\n",up1/t,down1/t);

}

int main()

{

long up1,up2,down1,down2;

char flag;

while(~scanf("%ld/%ld%c%ld/%ld",&up1,&down1,&flag,&up2,&down2))

{

count(flag,up1,up2,down1,down2);

}

return 0;

}

//

答：完成度不足，表现在遇到负数就完蛋。。。

#include <stdio.h>

struct fraction{

long int numerator;

long int denominator;

};

struct fraction add(struct fraction,struct fraction);

struct fraction sub(struct fraction,struct fraction);

struct fraction mult(struct fraction,struct fraction);

struct fraction divis(struct fraction,struct fraction);

struct fraction yuefen(struct fraction);

int main(){

struct fraction a,b,result;

char opCh;

scanf("%ld/%ld%c%ld/%ld",&a.numerator,&a.denominator,&opCh,&b.numerator,&b.denominator);

switch(opCh){

case '+':

result=add(a,b);

break;

case '-':

result=sub(a,b);

break;

case '\*':

result=mult(a,b);

break;

case '/':

result=divis(a,b);

break;

}

printf("%ld/%ld\n",result.numerator,result.denominator);

return 0;

}

struct fraction add(struct fraction x,struct fraction y){

struct fraction result;

result.numerator=(x.numerator\*y.denominator)+(x.denominator\*y.numerator);

result.denominator=x.denominator\*y.denominator;

result = yuefen(result);

return result;

}

struct fraction sub(struct fraction x,struct fraction y){

struct fraction result;

result.numerator=(x.numerator\*y.denominator)-(x.denominator\*y.numerator);

result.denominator=x.denominator\*y.denominator;

result = yuefen(result);

return result;

}

struct fraction mult(struct fraction x,struct fraction y){

struct fraction result;

result.numerator=x.numerator\*y.numerator;

result.denominator=x.denominator\*y.denominator;

result = yuefen(result);

return result;

}

struct fraction divis(struct fraction x,struct fraction y){

struct fraction result;

result.numerator=x.numerator\*y.denominator;

result.denominator=x.denominator\*y.numerator;

result = yuefen(result);

return result;

}

struct fraction yuefen(struct fraction input){

struct fraction result;

long int x,y,temp;

if(input.numerator>input.denominator){

x=input.numerator;

y=input.denominator;

}else{

y=input.numerator;

x=input.denominator;

}

while(y){

temp=x%y;

x=y;

y=temp;

}

result.numerator=input.numerator/x;

result.denominator=input.denominator/x;

return result;

}

2、编写函数，将单链表进行逆序，即表头变表尾，表尾变表头(15分)其中：

节点定义为：

structnode

{

intnum,structnode\*next

};

函数原型为：

voidturn(structnode\*head)

//

链表逆

#include<stdio.h>

#include<malloc.h>

struct node

{

int num;

struct node\*next;

}\*head;

void turn(struct node\* head)

{

//head为头结点无数据，采用头插法逆

struct node\*p=head->next,\*q;

head->next=NULL;

while(p)

{

q=p->next;

p->next=head->next;

head->next=p;

p=q;

}

//输出测试

while(head->next)

{

printf("%d\n",head->next->num);

head=head->next;

}

}

int main()

{

int i;

struct node \*p,\*q;

head=malloc(sizeof(struct node));

head->next=NULL;

//初始化数据

for(i=5;i>=0;i--)

{

p=malloc(sizeof(struct node));

p->num=i;

p->next=head->next;

head->next=p;

}

turn(head);

return 0;

}

//

答：我做了好久，怎么都觉得这个题由于函数原型只传递了链表头，且没有返回值，所以不能采用改变链表节点顺序的方法（最容易想到的头插法和尾插法），应该逆序链表中每个节点保存的元素值。

然后发现，其实完全不用那么麻烦，**只要单链表是有头节点的**。这题就是常规题了。常规带头节点的单链表逆置算法请翻阅各大数据结构教材。我这把自己犯二的记录贴上吧，不想看的可以略过。

这里我采用的方法是用头插法建立一个新链表，并复制元素。典型的空间换时间法，时间复杂度为3n，空间复杂度为n

应该有更高效的办法，就是从头节点后面开始用头插法建立链表，头尾节点单独赋值。这种算法就请别的高人写吧。我想到哪写到哪。

void turn(struct node \*head){

struct node \*newHead,\*p,\*q;

if(!head)//若表为空，不执行任何操作

return;

//初始化新表头

newHead=(struct node\*)malloc(sizeof(struct node));

newHead->num=head->num;

newHead->next=NULL;

//新表赋值为与旧表顺序相反

p=head->next;

while(p){

q=(struct node\*)malloc(sizeof(struct node));

q->num=p->num;

q->next=newHead;

newHead=q;

p=p->next;

}

//将旧表元素值变为与新表相同

p=head;

q=newHead;

while(p){

p->num=q->num;

p=p->next;

q=q->next;

}

//释放新表所占据的内存空间

p=newHead;

while(p){

q=p;

p=p->next;

free(q);

}

}

3、编写完整程序：

接收从键盘输入的仅由数字字符构成的字符串(假设字符串的最大长度为50)，统计并输出每个数字(0-9)的重复次数。(8分)

//

统计数字个数

#include<stdio.h>

#include<string.h>

int main()

{

int a[10]={0},i;//全部初始化为0

char s[50];

while(~scanf("%s",s))

{

int len=strlen(s);

for(i=0;i<10;i++) a[i]=0;//初始化

for(i=0;i<len;i++)

{

a[s[i]-'0']++;

}

for(i=0;i<10;i++)

printf("%d-%d\n",i,a[i]);

}

return 0;

}

//

答：

#include <stdio.h>

#define MAX 50

int main(){

char str[MAX];

char \*p;

int count[10]={0};

scanf("%s", str);

p=str;

while( \*p != '\0' ){

++count[\*p-'0'];

++p;

}

printf("数字0出现%d次\n", count[0]);

printf("数字1出现%d次\n", count[1]);

printf("数字2出现%d次\n", count[2]);

printf("数字3出现%d次\n", count[3]);

printf("数字4出现%d次\n", count[4]);

printf("数字5出现%d次\n", count[5]);

printf("数字6出现%d次\n", count[6]);

printf("数字7出现%d次\n", count[7]);

printf("数字8出现%d次\n", count[8]);

printf("数字9出现%d次\n", count[9]);

return 0;

}

4、编写完整程序，采用结构数组和指向结构的指针，接收输入的100个学生信息(包括学号和C语言课程期末总成绩)，输出最高、最低成绩和分别对应的学号(可能有多个同学都是最高分，可能有多个同学都是最低分)。(12分)

统计学生成绩

#include<stdio.h>

#define N 3

typedef struct

{

char num[10];

int cscore;

}\*sPoint,stu;

stu s[N];

void order(sPoint p,int \*max,int \*min)

{

int i;

\*max=p->cscore;

\*min=p->cscore;

for(i=0;i<N;i++)

{

if((p+i)->cscore>\*max)

\*max=(p+i)->cscore;

if((p+i)->cscore<\*min)

\*min=(p+i)->cscore;

}

}

void print(sPoint p,int max,int min)

{

int i;

printf("最高分\n");

for(i=0;i<N;i++)

{

if((p+i)->cscore==max)

printf("%s-%d\n",p[i].num,(p+i)->cscore);

}

printf("最低分\n");

for(i=0;i<N;i++)

{

if((p+i)->cscore==min)

printf("%s-%d\n",p[i].num,(p+i)->cscore);

}

}

int main()

{

int i,max,min;

sPoint p=s;

for(i=0;i<N;i++)

scanf("%s %d",(p+i)->num,&(p+i)->cscore);

order(p,&max,&min);

print(p,max,min);

return 0;

}

//

答：算法性能不好，求高性能算法！！

#include <stdio.h>

#define MAX 100

struct stuInfo{

int id;

int score;

};

int main(){

struct stuInfo students[MAX],\*p;

int max,min,i;

//输入学生学号和成绩

for(i=0;i<MAX; ++i){

scanf("%d %d", &students[i].id, &students[i].score);

}

//找出最大最小成绩值

max=min=students[0].score;

p=students+1;

for(i=1;i<MAX;++i,++p){

if(max<p->score)

max=p->score;

if(min>p->score)

min=p->score;

}

//输出成绩最高的学生

p=students;

for(i=0;i<MAX;++i,++p){

if(p->score==max)

printf("成绩最高者，学号：%d；分数：%d\n", p->id, p->score);

}

//输出成绩最低的学生

p=students;

for(i=0;i<MAX;++i,++p){

if(p->score==min)

printf("成绩最低者，学号：%d；分数：%d\n", p->id, p->score);

}

return 0;

}

四、从程序执行效率方面考虑，请简述C语言采取的一些措施和原因。(15分)

1：指针，可以操作内存

2：位移，可以作位运算

3：API，可以调用系统API，接近底层

4：宏define可以编译的时候替换

答：这是@李-互联网 通过伟大的百度百科搜出来的，让我们感谢他们吧！

1.使用指针：对于指针的理解简单点可以认为类似于汇编中的寻址方式，正是指针的存在使C语言威力无穷。有些程序用其他语言也可以实现，但C能够更有效地实现；有些程序无法用其它语言实现，如直接访问硬件，但C却可以。正因为指针可以拥有类似于汇编的寻址方式，所以可以使程序更高效。

2.使用宏函数：函数和宏函数的区别就在于，宏函数占用了大量的空间，而函数占用了时间。函数调用是要使用系统的栈来保存数据的，如果编译器里有栈检查选项，一般在函数的头会嵌入一些汇编语句对当前栈进行检查；同时，CPU也要在函数调用时保存和恢复当前的现场，进行压栈和弹栈操作，所以，函数调用需要一些CPU时间。而宏函数不存在这个问题。宏函数仅仅作为预先写好的代码嵌入到当前程序，不会产生函数调用，所以仅仅是占用了空间，而使程序可以高效运行。在频繁调用同一个宏函数的时候，该现象尤其突出。

3.使用位操作：位操作可以减少除法和取模的运算。在计算机程序中数据的位是可以操作的最小数据单位，理论上可以用"位运算"来完成所有的运算和操作。一般的位操作是用来控制硬件的，或者做数据变换使用，但是，灵活的位操作可以有效地提高程序运行的效率。

4.循环嵌套中将较长循环设为内存循环，较短循环设为外置循环，以减少cpu跨切循环层的次数，提高程序的运行效率。

再有一个@seven同学给出的简单版，请记忆关键字。再次感谢

1：指针，可以操作内存

2：位移，可以作位运算

3：API，可以调用系统API，接近底层

4：宏define可以编译的时候替换