指针和字符数组的概念和应用编程

一、指针概念-选做题（Concept of Pointer -Optional）

A. 指针定义和简单应用

1	2	3	4	5	6	7	8	9	10
D	**D**	**B**	**B**	**D**	**D**	**B**	**C**	**A**	**C**
11	12	13	14	15	16	17	18	19	20
C	**C**	**A**	**A**	**D**	**D**	**C**	**D**	**B**	**C**

B、 str[]和*str 的区别

答案：

0

0

1

1

C）指针和分配的空间

答案：

这段程序运行会出现异常。原因在于分配 **str** 指针的时候仅仅给了 **1** 字节的空间，但是 **strcpy** 拷贝了 **6** 字节到 **str**。运行输出后程序因为访问了没有分配的空间，当然崩溃了。如果 **strcpy(str,"");** 那程序是可以正常运行的。

二、指针实验 I 合并字符串（PointerExperiment I–MergeStrings）

（1） mergStrlen = len1 + len2;

（2） mergStr = new char[mergStrlen + 1];

（3） strcpy(mergStr,str1);

（4） char *tempStr = mergStr + len1;

（5） strcpy(tempStr, str2);

（6） delete [] mergStr;

特别说明： (4) 和 (5)可以合并为一个语句来完成：

strcpy(mergStr + len1, str2); 或 strcat(mergStr,str2);

三、指针实验 II 查找字符串

```
#include <iostream>
#include <cstring>
using namespace std;
char* myStrstr(char* str1,char* str2);
int main()
{
	chardest[10]="abcdabc";
	char* rp;
	char ch1[]= "c";
	char str2[]= "cda";

	rp=myStrstr(dest, ch1);

	if(rp==NULL)
		cout << "no" << ch1 << " exist" <<endl;
	else
		cout << "substring is" << rp <<endl;

	rp=myStrstr(dest, str2);

	if(rp==NULL)
		cout << "no" << ch1 << " exist" <<endl;
	else
		cout << "substring is" << rp <<endl;

	return 0;
}
// 参考答案
/*
char* myStrstr(char* str1,char* str2)，
找出 str2 字符串在 str1 字符串中第一次出现的位置(不包括 str2 的串
结束符) ， 返回该位置的指针,如 找不到,返回空指针。
*/
char* myStrstr(char* str1,char* str2)
{
	int i = 0;
```

```
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    if(len1 < len2)  //如果 str2 比 str1 长，返回 NULL
        return NULL;
    for(int i = 0; i< len1-len2+1; i++)     // 从 str1 的第一个字符位置开始测试,
    {
        int j;
        for(j=0;j<len2;j++)  // 判断 str2 是否与 str1 从下标 i 开始的子串相等
            if( str1[i+j] != str2[j])
                break;
         // 此时 str2 是 str1 的子串，该子串第一个元素在 str1 中的下标是 i
        if(j == len2)
            return &str1[i];
    }
    return NULL;     // 以上如果每个位置测试均失败，则返回 NULL
}
```

四、指针实验 III 计算子串

```
#include <iostream>
#include <cstring>
using namespace std;

char* mysubstr(char* srcstr, int offset, int length);

int main()
{
    char srcstr[] = "this is a test string!";

    char* tmp = mysubstr(srcstr, 16, 8);
    cout << "the substring is: " << tmp << endl;
    delete tmp;
    return 0;
}

// 参考答案
char* mysubstr(char* srcstr, int offset, int length)
{
    char *p;
    int srclen = strlen(srcstr);          // 先求出原字符串的长度

    if(offset < 0 || offset > srclen || length < 0) //   判断参数是否合理
        return NULL;
```

```
    if (offset + length > srclen)
        //若参数中目标子串长度 length 过长，调整一下目标子串长度
        length = srclen - offset;

    p = new char[length+1];  //申请一个动态数组，用指针 p 记录该数组地址

    int i = 0;
    for ( ; i < length; i++)
        p[i] = srcstr[offset + i]; //将指定位置的数组元素依次拷贝到动态数组中

    p[i] = '\0';      //  设置字符串结束字符

    return p;         //返回该动态数组的地址
}
```

五、指针和数组的应用

```
#include <stdlib.h>
#include <iostream>
#include <time.h>
#include <assert.h>
using namespace std;

int * getPolynomial (int num);
/*  函数在堆中申请一个动态数组用于存储最高次项为 num 的多项式，数组元素
用来表示多项式的系数。为 简单期间假定系数均为 int 类型，数组第一个元素
用于表示多项式的最高次项的次数，从第二项开始依次存储常数项，一次项系
数，…，最后一个元素是最高次项系数。
函数返回一个指向该动态 数组的指针。
*/

void setPolyCoef(int *pt, int maxval);
//  函数对多项式系数（pt 所指向的动态数组的元素)赋值
//  每个元素的值为在 0~ maxval-1 之间的随机数。

void releaseMem(int *pt);
//  函数用于释放由 getPolynomial 所申请的内存空间。

void printPolynomial(int *pt );
//  函数按照多项式的格式，从左到右依次打印高次项，…，常数项。
//  例如打印四次多项式：  5x^4 + 23x^3 + 105x + 2

double polynomialVal (int *pt, double x);
//  当自变量 x 的值已知时，求多项式的函数值
```

```
int *polynomialAdd(int *pt1, int *pt2);
// 求两个多项式 pt1, pt2 的和，返回指向（求和）结果的指针。

int main()
{
    int *p1, *p2, *p3, num1=7, num2=5;
    int maxval =50;

    srand(time(NULL)); //设置随机数的种子

    p1= getPolynomial(num1);
    setPolyCoef(p1, maxval);
    printPolynomial(p1);

    double x=1.2;
    cout << "the polynomial p1's value when x = " << x <<      " is: "
        << polynomialVal (p1, x) << endl;

    p2= getPolynomial(num2);
    setPolyCoef(p2, maxval);
    printPolynomial (p2);

    p3 = polynomialAdd (p1, p2);
    printPolynomial (p3);

    cout << "the polynomial p3's value when x = " << x <<      " is: "
        << polynomialVal (p3, x) << endl;

    releaseMem(p1);
    releaseMem(p2);
    releaseMem(p3);

    return 0;
}

// 参考答案
int * getPolynomial (int num)
{
    int *p;
    p = new int[num+2];   // 存放多项式的次数、各项的系数
    p[0] = num;   // 设置多项式的次数
    return p;
}
```

```
void setPolyCoef(int *pt, int maxval)
{
	int n;
	n = pt[0];	//先查询多项式的次数

	for(int i=1; i<= n+1; i++)
		pt[i] = rand()*maxval/(RAND_MAX + 1);
}

void releaseMem(int *pt)
{
	delete [] pt;
}

bool remainItem(int *pt, int start);
// 此辅助函数用来判断从系数数组中下标从 start 到 1，
// 是否还有系数不为 0 的项,用于打印时的格式控制

void printPolynomial(int *pt )
{
	int n = pt[0];  // 先查询多项式的次数
	cout << endl;
	for(int i = n+1; i > 2; i--) // 依次打印 n 次项，..,到二次项。
	{
		if(pt[i])
		{
			cout << pt[i] << "x^" << i-1;
			if(remainItem(pt,i-1))
				cout << " + ";
		}
	}

	if(pt[2])
	{
		cout << pt[2] << "x ";		// 一次项
		if(pt[1])
			cout << " + ";
	}

	if(pt[1])
		cout << pt[1];		// 常数项
	cout << endl;
}
```

```
// 此函数用来判断从系数数组中下标从 start 到 1，是否还有系数不为 0 的项
bool remainItem(int *pt, int start)
{
	while(start>=1)
	{
		if(pt[start])
			return true;
		else
			start--;
	}
	return false;
}

double pow(double x, int num);		//  此辅助函数计算 x 的方幂 x^num

double polynomialVal (int *pt, double x)
// 当自变量 x 的值已知时，求多项式的函数值
{
	double val=0; // 设置多项式的初值为 0
	int n = pt[0];  // 先查询多项式的次数

	for(int i = 0; i <= n; i++)		// 依次计算常数项，..,到 n 次项。
		if(pt[i+1])
			val = val + pt[i+1] *	pow(x,i);

	return val;
}

double pow(double x, int num)		//  此辅助函数计算 x 的方幂 x^num
{
	double pv=1;
	for(int i=0; i < num; i++)
		pv = pv * x;
	return pv;
}

//求两个多项式 pt1, pt2 的和，返回指向（求和）结果的指针。
int *polynomialAdd(int *pt1, int *pt2)
{
	int *p, num, mini, i;

	num = pt1[0] > pt2[0] ? pt1[0] : pt2[0];	//找到两个次数中较大的
	// 注意：调用该函数将得到一个存储多项式次数和各项系数的动态数组
```

```
    p= getPolynomial(num);
    mini = pt1[0] > pt2[0] ? pt2[0] : pt1[0];    //找到两个次数中较小的
    for(i=1; i<= mini+1;i++)            //先把有同类项部分相加
    {
        p[i] = pt1[i] + pt2[i];
    }

    if(pt1[0] > pt2[0])        // 将 pt1 剩下的高次项系数赋值到 p
    {
        while(i <= num + 1)
        {
            p[i] = pt1[i];
            i++;
        }
    }
    else //   将 pt2 剩下的高次项系数赋值到 p
    {
        while(i <= num + 1)
        {
            p[i] = pt2[i];
            i++;
        }
    }
    return p;
}
```