**《程序设计专题》课程期末模拟考试试卷**

课程号： 211G0260 ，开课学院： 计算机学院\_\_

考试试卷：√A卷、B卷（请在选定项上打√）

考试形式：√闭、开卷（请在选定项上打√），允许带 ∕入场

考试日期： 2018 年 06月 19 日,考试时间： 120 分钟

**诚信考试，沉着应考，杜绝违纪.**

**考生姓名： 学号： 所属院系： \_**

**(注意：答题内容必须写在答题卷上，写在本试题卷上无效)**

试题一、单选题（每小题3分，共30分）

1. 在\_\_\_\_\_\_\_，指令#include <stdio.h>会被替换为文件stdio.h的内容.

|  |  |
| --- | --- |
| A．编辑代码时 | B．运行程序时 |
| C．编译预处理时 | D．程序连接时 |

1. 把数组int a[10][20]传入函数aver求平均值，不正确的代码段\_\_\_\_\_\_\_.

|  |  |  |
| --- | --- | --- |
| A．int a[10][20] = {1,2,3,4,5};  aver(a, 200);  float aver(int \*a, int size)  { /\*…\*/ } | B．int a[10][20] = {1,2,3,4,5};  aver(a, 10);  float aver(int (\*a)[20], int rows)  { /\*…\*/ } | |
| C．int a[10][20] = {1,2,3,4,5};  aver(\*a, 200);  float aver(int \*a, int size)  { /\*…\*/ } | | D．int a[10][20] = {1,2,3,4,5};  aver(a, 10);  float aver(int a[][20], int rows)  { /\*…\*/ } | |

1. 关于递归函数调用的说法，错误的是\_\_\_\_\_\_\_.

|  |
| --- |
| A. 每次调用函数时分配参数和局部变量的存储空间，退出函数时释放； |
| B．随着递归函数的层层深入，存储空间的一端逐渐增加，然后随着函数调用的层层返回，存储空间的这一端又逐渐缩短； |
| C．递归存在着可用堆栈空间过度使用的危险； |
| D．递归调用的层数可以是无限制的。 |

1. 已知sort()函数原型为：void sort(int a[], int size)；，并且另有fun()函数，它的第一个参数为整型指针，第二个参数可以接受函数sort的入口地址，则函数fun的函数原型为\_\_\_\_\_\_\_.

A. void fun(int \*p, void(\*p)(int a[], int size));

B．void fun(int \*p, void\* q);

C．void fun(int \*p, void(\*q)(int a[], int size));

D．void fun(int \*p, void(\*q)(int a[size]);

1. 定义宏MAX，求两个数的最大值，最合适的是\_\_\_\_\_\_\_.

A．#define MAX(a,b) (a>b)?a:b

B．#define MAX (a,b) (a>b)?a:b

C．#define MAX(a,b) ((a)>(b)?(a):(b))

D．#define MAX (a,b) (((a)>(b))?(a):(b))；

1. 在下面的代码中，p2是一个\_\_\_\_\_\_\_.

typedef int \*ptr;

ptr p1, p2;

|  |  |
| --- | --- |
| A．整数 | B．指向整数的指针 |
| C．代码有语法错误 | D．以上都不对 |

1. 假设高考考生成绩与简单信息的结构类型定义如下：

struct scoretype {

int Math, Chinese, English, Integrated, Total;

};

struct student {

char name[20];

unsigned long num; // 准考证号码

unsigned long rank; // 排名序号

struct scoretype score;

} s[100]，\*p;

则”数学Math成绩超过140分的同学总分Total额外再加5分”的正确写法是\_\_\_\_\_\_\_.

|  |
| --- |
| A. if(p-> score->Math>140) p-> score->Total +=5; |
| B．if(\*p-> score.Math>140) \*p-> score.Total +=5; |
| C．if(\*p. score.Math>140) \*p.score.Total +=5; |
| D．if(p-> score.Math>140) (\*p). score.Total +=5; |

1. 假设单链表的结点结构定义如下:

struct node {

int data; struct node \*next;

} \*p, \*n;

并且假设当前p指向单链表中的某内部结点（既非首结点也非尾结点），删除该结点的正确语句序列是\_\_\_\_\_\_\_.

|  |
| --- |
| A. n=p->next; p->next=n->next; free(n); |
| B．n=p->next; p->data=n->data; free(n); |
| C．n=p->next; p->next=n->next; p->data=n->data; free(p+1); |
| D．n=p->next; p->data=n->data; p->next =p->next->next; free(n); |

1. 在最坏情况下，选择排序算法的时间复杂度是\_\_\_\_\_\_\_.

|  |  |
| --- | --- |
| A．O( *N* ) | B．O(*N*2) |
| C．O( 2*N*) | D．O( *N*3 ) |

1. 若归并非递归排序算法思想为：

第一趟归并：以间隔为1的进行归并，也就是说，第一个元素跟第二个进行归并。第三个与第四个进行归并；

第二趟归并：以间隔为2的进行归并，[(1) (2)]与[(3) (4)]进行归并，[(5) (6)]与[(7) (8)]进行归并；

第三趟归并：以间隔为2\*2的进行归并；

同理，直到2k超过数组长度为止。

则按照上述算法，对 4 3 7 8 0 9 2 1 5 6 进行排序。第二趟归并后的结果是\_\_\_\_\_\_\_.

|  |  |
| --- | --- |
| A．3 4 7 8 0 1 2 9 5 6 | B．0 1 2 3 4 7 8 9 5 6 |
| C．3 4 7 0 8 9 2 1 5 6 | D．3 4 0 7 8 9 2 1 5 6 |

试题二、改错题（每小题3分，共18分）

1. 找出并更正以下结构定义中的错误.

struct StudentListNode {

int num, score;

StudentListNode \* next;

};

1. 以下的递归函数将任意十进制整数num转换成base进制数(1<base<10)输出，找出错误并修改正确。

void change(int num, int base) {

if(num/base>0) {

printf("%d", num % base);

change(num / base, base);

}

else printf("%d", num);

}

1. 以下程序使用了条件编译，本次运行想输出“BBBB”，找出错误并修改正确。

#include <stdio.h>

#define DEBUG 0

int main(void ){

#ifdef DEBUG

printf("AAAA\n");

#else

printf("BBBB\n");

#endif

return 0;  
 }

1. 函数Find在数组x中查找值为v的元素，并返回元素的下标。如果满足要求的元素不存在，那么返回-1。 假设参数x的元素按照升序排序。找出错误并修改正确。

int Find(float x[], int n, float v) {

int m = n/2; /\*数组中间下标\*/

if( n<1 ) return (-1);

if( x[m]<v ) return Find(x+m+1, n, v);

else

if( x[m]>v ) return Find(x, m-1, v);

else return m;

}

1. 以下程序把结构变量a赋值给b，再输出结构变量b，结果输出的b.name是乱码，而b.score是正常的。找出错误并修改正确。

typedef struct student {

char \*name;

int score;

} ST;

void main() {

ST a, b;

a.name = malloc(100);

scanf("%s %d", a.name, &a.score);

b = a;

free(a.name);

printf("%s %d\n", b.name, b.score);

}

1. 函数buildlist()用来创建一个双向链表，main()在调用buildlist()后，按逆序输出链表中的各个节点: 4 3 2 1 0。已知函数buildlist()中有一个语句有逻辑错误，找出该语句并修改正确。

#include <stdio.h>

typedef struct list {

int data;

struct list \*prev;

struct list \*next;

} LIST;

LIST \* buildlist(int n) {

int i;

LIST \*phead = NULL, \*pfore = NULL, \*p;

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

p = malloc(sizeof(LIST));

p->data = i;

p->next = NULL;

p->prev = pfore;

pfore -> next = p;

if(phead == NULL)

phead = p;

pfore = p;

}

return phead;

}

void main() {

LIST \*p;

p = buildlist(5);

while(p->next != NULL)

p = p->next;

while(p != NULL) {

printf("%d ", p->data);

p = p->prev;

}

}

试题三、问答题（共10分）

1. 简述全局变量、静态全局变量、静态局部变量的异同。（6分）
2. 假定解决一个问题有两种算法程序，采用算法A的程序A和采用算法B的程序B，它们在某同一计算机上编译运行，已知问题的规模n和解决问题两个程序的运行时间T(A)和T(B)见下表：（us表示微秒，ms表示毫秒）（4分）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | 1,000 | 2,000 | 4,000 | 8,000 |
| T(A) | 100 us | 410 us | 1.62 ms | 6.52 ms |
| T(B) | 1.1 ms | 2.2 ms | 4.41 ms | 8.83 ms |

请估计算法A与算法B的时间复杂度。

试题四、程序填空题（每空2分，共24分）

1. 函数f()用来计算Fibonacci数列(1、1、2、3、5、8、...)的第n项，函数g()用来把Fibonacci的前n项填入数组a中，函数main()输出Fibonacci的前n项。其中f()、g()均为递归函数。

int f(int n) {

if(n==1 || n==2) return 1;

return 　 （1） 　 ;

}

void g(int a[], int n) {

if(n>=2) {

g(a, n-1);

a[n-1] = f(n);

}

else

　 （2） 　 ;

}

void main() {

int a[100], i, n;

scanf("%d", &n);

　 （3） 　 ;

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

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

}

1. 一群猴子要选新猴王。新猴王的选择方法是：让n只候选猴子围成一圈，从某位置起顺序编号为1~n号。从第1号开始报数（从1到3），凡报到3的猴子即退出圈子，接着又从紧邻的下一只猴子开始同样的报数。如此不断循环，最后剩下的一只猴子就选为猴王。在空缺处填上正确的内容。

#include <stdio.h>

#define M 10

int find\_next( int start, int monkey[ ][M], int n) {

int i=(start+1)%n;

while( monkey[1][i]==0 )

　 　 （4） 　 ; 　 ；

return i;

}

int main( ) {

static int monkey[2][M]; /\* monkey[0] 存储编号,monkey[1] 表示是否在列\*/

int i, count, n, start=0;

scanf("%d",&n);

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

monkey[0][i] = i+1;

monkey[1][i] = 1; /\* all the monkeies are in the circle \*/

}

for( count=n; count>1; count-- ) {

i = 　 （5） 　 ;

i = find\_next(i,monkey,n);

　 （6） 　 ;

start = find\_next(i,monkey,n);

}

printf("The king is monkey[%d].\n",monkey[0][start]);

return 0;

}

1. 以下是完成快速排序的函数，试在空缺处填入适当的内容。

void quicksort(int list[],int m,int n)

{

int key,i,j,k;

if( m < n) {

k =(m+n)/2;

swap(&list[m],&list[k]);

key = list[m];

i = m+1;

j = n;

while(i <= j) {

while((i <= n) && (list[i] <= key) i++;

while(　 （7） 　 ) j--;

if(　 （8） 　 ) swap(&list[i],&list[j]);

}

　 （9） 　 ;

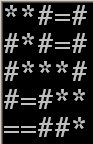
quicksort(list,m,j-1);

quicksort(list,j+1,n);

}

}

1. 函数main()中，定义了迷宫地图map，其中'='表示道路，'#'表示墙；数组mark用来标记map中对应位置是否已走过，走过标为1，没走过则为0；begin是走迷宫的起始位置0行0列，end表示迷宫的终点4行4列。函数dig使用递归法走迷宫，其中参数rows为迷宫的行数，cols是迷宫的列数；当走到终点时dig()返回1，走不通时返回0。dig()的递归过程如下:①先判断begin能否走，例如该位置已经走过或者坐标越界就不能走，不能走就返回0；②把begin设为已走过，地图对应位置标成'\*'；③以begin为中心，按左上右下4个方向尝试对它周围的4个位置进行dig()；④若4个方向均走不通，则放弃begin这个点，回到begin前面那个点。程序的输出结果如下：



typedef struct point {

int row, col;

} POS;

int dig(char \*map[5], char mark[5][5], int rows, int cols, POS begin, POS end) {

int k, d[4][2]={{0,-1}, {-1, 0}, {0, 1}, {1,0}};

int r=begin.row, c=begin.col;

if(r < 0 || r >=rows || c < 0 || c >=cols) return 0;

if(　 （10） 　 ) return 0;

if(mark[r][c] == 1) return 0;

mark[r][c] = 1;

map[r][c] = '\*';

if(r == end.row && c == end.col) return 1;

for(k=0; k<4; k++) {

　 （11） 　 ;

begin.col = c+d[k][1];

if(dig(map, mark, rows, cols, begin, end) == 1) return 1;

}

　 （12） 　 ;

return 0;

}

void main() {

char \*map[5] = {

"==#=#",

"#=#=#",

"#===#",

"#=#==",

"==##="

};

static char mark[5][5];

POS begin={0,0}, end={4,4};

int i;

dig(map, mark, 5, 5, begin, end);

for(i=0; i<5; i++) {

puts(map[i]);

}

}

试题五、算法设计（共18分）

1．通过malloc()函数动态申请二维数组所需的存储空间，采用如下方法来生成具有m行n列的动态大小的二维整型数组(m与n为变量，其值要在运行时确定)：

int \*\*pa = (int\*\*)malloc(m\*sizeof(int\*)); /\*生成动态数组，该数组具有m个int类型的

指针分量\*/

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

pa[i] = (int\*)malloc(n\*sizeof(int)); /\*pa[i]为第i行首地址，为每行分配了n个

int型存储空间\*/

动态申请三个二维整型数组A、B、C，其中A为m行n列，B为n行m列，C为m行m列，正整数m和n由用户从键盘输入。

（1）实现输入一个二维整型数组的函数，原型为：void input(int \*\*pa, int row, int col)；(4分)

（2）实现矩阵A与B相乘的函数，原型为：void mmult(int \*\*pc, int \*\*pa, int row,int col, int \*\*pb); (4分)

（3）（2）中相乘算法的时间复杂度是多少？ (2分)

2．下面是一个由黑白格子构成的正方形网格图。在图中，如果两个网格都是黑色的，并且能够从一个网格连续移动到另一个网格，则称这两个网格属于一个块。（移动可以按照水平、竖直或对角线方向，移动过程中只能经过黑色格子。）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

假设已有C语言程序：

|  |
| --- |
| #define BLACK 1 /\* 黑色方格 \*/  #define WHITE 0 /\* 白色方格 \*/  #define HEIGHT 8  #define WIDTH 6/\* 宽度为6，高度为8的图 \*/  #include .... /\* 头文件已正确包含 \*/  void generateRandomMap( int map[HEIGHT][WIDTH]);  int map[HEIGHT][WIDTH];  void eraseBlob(int row, int col);  int main(void) {  int row, int col;  generateRandomMap(map);  scanf("%d%d", &row, &col);  eraseBlob(row, col);  return 0;  }  void generateRandomMap( int map[HEIGHT][WIDTH])  { /\*...\*/}  void eraseBlob(int row, int col)  { /\*...\*/} |

其中，二维的黑白网格图用数组map表示，当表示黑色网格时，元素值为1；表示白色网格时，元素值为0。generateRandomMap()函数可以在map数组中生成一个随机的黑白网格图。eraseBlob()函数会将map[row][col]所在的块变为白色。例如，当map表示上图所示的网格图时，调用eraseBlob(2, 2)后，map变为下图：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

（1）简要描述eraseBlob()函数的实现思路；(4分）

（2）写出该函数的定义。(4分）