C 语言部分

未解之题

1. 利用数组处理批量数据，Pro 提升选择题 16 题?

1.程序设计与 C 语言

1. C 语言是一种通用的、高级的编程语言,具有很好的移植性,能够在不同的操作系统下运行。
2. .exe 结尾的也是一种二进制文件，.obj 结尾的不能直接执行
3. C 语言中，C 语言程序的一个函数就是一个程序模块
4. 翻译程序有解释和编译两种方式
5. C 语言共有 32 个关键字，34 种运算符
6. C 语言中 main 函数的位置可以任意
7. 16 进制的数组，前面 0x 和 0X 开头都可以
8. float，double 都可以使用++,--
9. 变量被定义后，如果没有被赋值，那么将不会有一个确定的本身值

2.程序的存储和运算

0. **在计算机中储存整数是用的补码。**
1. 决定表达式计算顺序的两个因素是优先级和结合性
2. sqrt(x)：计算根号内 x
3. pow(x,y)：计算 x^y
4. exp(x)：求 e^x
5. 浮点数使用指数形式时，字符 e 之前必须需有数字，且 e 后面必须有指数，指数必须为整数
6. 一个实型变量的值肯定是精确的？由于实型变量在内存中的存储单元是由有限个字节组成的，能提供的数字也是有限的,在有效位以外的数字将被舍去,因此可能会产生误差
7. 在 C 语言中，实型变量分为三种类型，它们是单精度浮点型(float 型)，双精度浮点型(double 型)，长双精度浮点型(long double 型)
8. 字符串常量使用一对单引号界定单个字符，而字符串常量使用一对双引号来界定若干个字符序列
9. **转义字符\ddd 表示 8 进制转义字符，d 范围 0~7**
10. **转义字符\xhh 表示 16 进制转义字符，16 进制转义字符，h 范围 0_9，$_a$f，A~F，3 位 16 进制**
11. **float，double 类型没有 unsigned**
12. 宏定义不是语句，后面不可以加分号
13. 关于负数取余的，只要左边的为负数，则结果肯定带负数
14. (a++)++ 不合法,暂时认为 a++再额外加的是错误的
15.

```
/*
printf("%d\n", 5 % 3); 2
printf("%d\n", -5 % 3); -2
printf("%d\n", -5 % -3); -2
printf("%d\n", 5 % -3); 2
*/
```

16. a=(b=2*3,b*5),b+6;求 a 的值？a 为 30，第一个括号内输出 b*5=30,然后 a=的等于号优先级比 b+6 高，所以 a=30
17. a=6,b=4, ++a+b%a+++b = (a+1)+(b%a) +b = (6+1) + 4%6 + 4 = 15，即变量在计算过程中不会前面加值后，自动使用该变量的最新值?
18. C 语言中字符型数据在内存中的存储形式是 ASCII
19. char 的范围是-128~127，若赋值是数的超过 127 时，就减去 256，低于-128 就加 256，这是因为变量溢出而得出的一个规律，题目中 256 超过 127，变量的值 256-256 为 0。
20. float 型输入按最普通的输入方式即可，不用规定表列，如(%3.2f)，规定表列输入 enter，会失效
21. 16 位机器下，unsigned int x = 0xffff; printf("%u\n",x); 打印 65535。解：题目中提到了 16 位机，int 类型为 2 个字节，若 x 定义成无符号的话，就不存在负数，0xffff 在内存存放以补码方式，1111 1111 1111 1111，无符号数，最高位不是符号位，所以该数就是最大值 65535
22. printf 输出负数会正常输出负数，输出的参数过多时会自动去除过多参数
23. 当 scanf 同时输入整数和浮点数时，scanf("%3d%f",&i,&x);时，前面 3 代表 3 个空，所以可以 0013.14,001 占满 3 个后，后面自动填充 3.14，所以 i=1,x=3.14
24. **分数在 C 语言中默认是用 double 存储的 占用 8 个字节**
25. scanf 可以将读取的数字字符转化成整数形式，gets 不能
26. 实数的表示形式分：1.十进制小数形式 2.指数形式，指数形式中 123E-8,123E+8，123E8 都是可以的
27. 打印 float f = 12.4565; printf("%.3f", f); 为 12.457 因为小数第三位靠第四位四舍五入，如果第四位为 4 的，则 12.457
28. char ch; scanf("%3c", &ch); 输入 XYZ 回车，最终打印是 X，因为不管 c 前面为多少位，最终只能获取第一位

3.顺序程序设计

1. 在 C 语言中，当表达式的值为 0 时,表示逻辑值“假”,当表达式的值为非 0 时,表示逻辑值“真”
2. 编译系统在表示逻辑运算结果时以数值 1 代表“真”，以 0 代表“假”。但在判断一个量是否为“真”时以 0 代表“假”，以非 0 代表“真”
3. scanf 函数中没有精确控制,例如： scanf(“%5.2f”， &a);是非法的.

4.选择结构程序设计

1. C 语言规定，else 子句总是与它上面最近的未配对的 if 配对
2. C 语言中对关系运算的结果，用数值 1 表示真，用 0 表示假
3. if 判断里面只要不为 0 就是真，即 if(-1)其实是 true
4.

```
/*
while(!x)中的!x与下面的()表达式等价。
A.x==0
B.x==1
C.x!=5
D.x!=0

类似这种题，判断条件都要为true，即!x == 1，那x==0，所以在没有!号的情况下只有(x==0)才符合题意，选A
 */
```

5.

```
/*
int i = 011; // 八进制
i++; // 八进制的运算都是转换成十进制之后再进行加减乘除
print("%d", i); // 打印9，011相当于十进制的8
 */
```

4. 算数表达式，赋值表达式的值不只 0 或 1，可以是一个确切的值
5. 逻辑表达式和关系表达式的值只有 0 或 1
6. 三目运算符中表达式三不可以单独出现赋值运算符，比如 a>b?5:b=5，这样，但是可以 a>b?5:(b=5)
7. 类似 x=y=z*2 是可以的
8. 逻辑表达式左右的判断条件是以非 0 和 0 来判断，非 0 代表真。例如-1 && -1 的值为 1

5. 循环结构程序设计

1. 当 for 语句省略条件表达式时,表示该条件一直为真
2. switch 如果命中后当前 case 没有 break，那会自动流入下一个判断条件,default 也会流入，不管下面的判断条件是否命中，直到遇到 遇到一个 break
3. switch case 的判断条件只能是字符型和整形
4. for 循环一旦在条件 2 断了，就不会执行条件 3

6.利用数组处理批量数据

1. 只要定义了数组，那不管数组有没有值，数组所占的大小都是固定的，

```
/*
int a[10] = {1,2,3,4}，若整形变量占4个字节，则数组在内存中占40个字节
*/
```

2. **数组的下标不能为变量，整型表达式中可以包含整形变量。所以数组的下标应该为整型常量表达式**
3. 如果没有给一个整型数组的元素赋初值,则其元素缺少的初值全部为 0
4. C 语言中只能逐个引用数组元素而不能一次引用整个数组。
5. 在 C 语言中，数组以按行为主顺序存放
6. strcpy(p1,p2) 是从 p1 的默认第 0 个位置开始复制，但是这个位置可以指定，比如 ch="123abcde"; strcpy(ch+4, "true")，结果是"123atrue"
7. char[20]说明 str 可以存放一个字符串，最多能表示 19 个字符，最后 1 个需要给'\0'
8.

```
/*
char str[20] = "ab\n\\0122\\\0"; 使用strlen(str)得出的结果是

拆分为 a,b,\n,0,1,2,2,\\,\0共10个，但是strlen不计算\0，所以是9
*/
```

7. C 语言声明数组的时候，如果不带大小的参数，必须初始化，比如不能 char str[];
8. 数据元素之所以相关,是因为它们具有相同的名字和类型
9. **C 语言中，数组在内存中占一片连续的存储区，由数组名代表它的首地址。数组名是一个地址常量，不能对它进行赋值运算**

7. 利用函数实现模块化

1. 在 C 语言中,形参必须是变量,实参可以是常量、变量或表达式。函效的形参可以有个,可以是相同类型,也可以是不同类型。实参与形参的类型应相同或赋值兼容。
2. 如果函数值的类型与返回值的类型不一致,则以函数值类型为准。
3. 在 C 语言中,每一个变量和函数都有两个属性,分别是数据类型和数据的存储类别。
4. void 型函数无返回值,无法进行运算,只能作为语句,故不可以出现在表达式中。
5. 如果被调用函数的定义出现在主调函数的数声明后面，则在主调函数中需要对其所调用的函数作函数声明
6. 用多维数组名作为函数的实参和形参,在被调用函数中对形参数组声明时可以指定每一维的大小,也可以省略第一维的大小说明,但是不能把第二维以及其他高维的大小说明省略

7. 函数内参数的命名不能和形参的名称一样，会报错
8. 若一个函数不需要形参,则在定义该函数时,应使形式参数表为空或放置一个 void
9. 当 return 语句被执行时,程序的执行流程无条件地从一个函数跳转到另一个函数。
10. C 语言中，若对函数类型未加说明，则函数的隐含类型为 int
11. **全局变量就是静态变量，比如可以在最外层 static char ch;**
12. double fun(int x,int y)是函数定义，double fun(intx, int y); 是函数声明(带分号)
13. C 语言中形参占存储单元，未调用时不占存储单元，一般情况是指调用的情况下
14. **函数内的静态变量只会在第一次调用的时候赋初值，第二次进来不会重新赋值**

8. 善于使用指针

1. 指针变量可以有空值,即“P=NULL;”,NULL 是一个符号常量,代表整数 0,它使 P 指向地址为 0 的单元。
2. 对于类型相同的两个指针变量,它们之间不能进行加(+)运算。
3. 数组名是常量，不能对其进行加减乘除，比如 int a[10]; a++是不允许的
4. 不能把二维数组赋值给指针变量
5. 指针和数组名其实都是一样，都有指向作用，比如：int arr[3] = {1,2,3}, *p=a; 如果输出 a[1]的话可以用 a[1],p[1], *(a+1),*(p+1)是等同的，只是如果 p 事先有移动的话就不同，可以理解为数组名的指向不能移动，而 p 可以移动
6. 如果有 int arr[3] = {1,2,3}, *p=a; 则 p=p+3 是错误的 a 只有 3 位，这样数组会越界
7. 数组指针

```
int a[3][4] = {{1, 3, 5, 7}, {6, 11, 13, 15}, {17, 19, 21, 23}};
int(*p)[4] = a; // 表示p是指针变量,指向有4个元素的一维数组,数组元素为整型，即指向a[0]的首地址
printf("%d\n", (*p)[1]); // 打印3 (*p)带括号情况下，先求*p也就是a[0][0]，然后(*p)[1]，即求a[0][1]
printf("%d\n", *p[1]); // 打印6 *p不带括号，根据优先级先求p[1]，p是指向a[0]的，则p[1]指向a[1]，然后再
```

8. 指针数组

```
char *str[] = {"abcd", "efgh", "ijkl"}; // 定义有3个指针的指针数组，数组的的第n项分别为数组的第n个，
printf("%s\n", str[1]); // efgh
printf("%c\n", str[1][0]); // e
printf("%s", *(str + 1)); // 输出efgh，因为str指向第一个，加1就是第二个，指向efgh
printf("%s", *(str + 1) + 1); // 输出fgh，为什么能在指向的变量基础上加1呢，因为*(str + 1)实质是指向一
printf("%c", *(*(str + 1) + 1));  // 输出f
```

9. 已知有程序片段"int i;float f;xxx;p=&i;p=&f;",若要保证该段程序没有任何语法错误,p 应当声明为 void *p;这样可以躲过基类型检查

10. float (*p)()代表初始化一个指向函数的指针，记得要加括号
11. 若 char a[30], *p=a; p="123"会成功，但是 a[30]里面实际赋值会出问题。要严格遵守 char a[10]和 char *a;两种格式的赋值方式
12. **int a[5] = {1, 2, 3, 4, 5}; 要想获取 a[1]的地址值，不能++a，数组名是常量，不能修改。但是可以 a+1, 因为这样只是生成一个临时变量存起来，对原来的 a 没有进行赋值操作**

结构体

1. 声明结构体时，系统并不为其分配存储空间，只有在结构体类型定义变量时猜对变量分配存储空间
2. 共用体类型可以出现在结构体类型定义中,也可以定义共用体数组。反之,结构体也可以出现在共用体类型定义中,数组也可以作为共用体的成员。
3. 共用体中只能对一个成员赋值
4. struct 是用户定义结构体类型的关键字 struct stu 是用户定义的结构体类型(stu 为类型名),stutype 是用户定义的 struct stu 类型的变量名,a 和 b 是用户定义的结构体成员名
5. 引用结构体变量中的成员的值，用到的运算符是**成员运算符(.)**
6. 在共用体变量中只能存放一个值,不能对它的多个成员同时赋值。比如 a={1,'z'}
7. 可以对共用体变量进行初始化,但初始化表中只能有一个常量.
8.

```
struct sk {
int a;
float b;
}data,*p;
// 若要使p指向 data 中的a域,正确的赋值语句是
// p = &data.a; 错误
// p = (struct sk*)&data.a; 正确

/*p是一个指向struct sk类型对象的指针变量,它用来指向一个struct sk类型的对象，不应用来指向某一结构体成员,
*/
```

利用文件保存数据

1. 写文件是指将数据信息从计算机内存存人磁盘,读文件是指将数据信息从磁盘调入计算机内存。
2. 文件的随机读写可用于二进制文件，也可用于文本文件
3. 函数 feof()用来判断文件的当前状态是否读完。若读完，则返回值为非 0，否则返回值为 0
4. 若需要使文件指针 fp 指到文件的末尾,可调用函数 fseek(fp, 0, 2);若需要使文件指针 fp 重新指向文件函数 rewind(fp);若要使文件指针 fp 指到距离文件开头 10 个字节的位置,可调用 fseek(fp, 10, 0)。
5. 关闭文件用 fclose 函数，成功执行关闭操作时，返回值为 0，否则返回 EOF(-1)
6. fseek 可以实现顺序读写，随机读写和改变文件位置标记
7. C 语言位置标记文件末尾，在最后一个字符的后面，而非最后一个字符

8. w：表示 fopen 文件时会清空掉原文件（如果存在）的信息，并重新写入，在不 fclose 文件的情况下，多次 fwrite 也是追加写入到文件末尾的，不会覆盖之前 fwrite 的内容。
9. a：表示 fopen 文件时会保留原文件（如果存在）的信息，并追加到末尾写入，每次 fwrite 写入到文件末尾。

数据结构部分

2.顺序表

1. 顺序表的合并，所谓的比较次数是，比如 a 和 b 合并，a 有 n 个，b 有 m 个，n 小于 m，b 并入 a，b 的一个值与 a 里面的每一个进行对比，即最少对比 n 次
2. 对一个空循环单链表,有 head->next= =head,推理 head->next->nexthead->nexthead。对含有 1 个元素的循环单链表,头结点(头指针 head 指示)的 next 城指向该唯一元素结点,该元素结点的 next 域指向头结点，因此也有 head-next==bead. 所以有可能 0 个或 1 个结点
3. 线性表除第一个结点外，每个结点都有一个直接前驱，**而不是每个结点都有一个直接前驱**
4. 在双向循环链表中插入一个新结点时，应修改 4 个指针域的值
5. 要想在第 i 个结点之前插人一个新结点,必须先找到第 i-1 个结点(即第 i 个结点的直接前驱),其时间复杂度为 O(n)
6. 线性表用数组表示，假定操作表中任意元素的概率相同，则操作一个元素平均需要移动的个数
    - 删除：(n-1)/2，因为最多移动 n-1 个，最少 0
    - 插入：(n+1)/2，因为最多移动 n 个，最少 0
7. 在一个单链表中删除 p 所指结点的后继结点时，应该执行 p->next=p->next->next
8. **创建一个包括 n 个结点的有序单链表的时间复杂度是 O(n^2)，因为要求有序，可能需要进行数据判断大小，如果为无序的话就是 O(n)**

3. 栈和队列

1. 利用栈后进先出的原则可以计算后缀表达式
2. 栈后进先出的特点就决定了栈可以对输入输出的序列部分或全局求逆。
3. 要判断是否与数据的存储结构有关,只需要看这种结构到底有没有具体使用到顺序存储或链式存储,如果有,那就一定和数据的存储结构有关。比如栈，只是逻辑结构，没有具体说是由顺序存储还是链式存储
4. 将一个递归算法改为对应的非递归算法时，通常需要使用栈，

4. 串，数组和广义表

1. 广义表可以是一个递归的表，即广义表也可以是其本身的一个子表
2. 由于存储单元是一维结构，而数组可能是多维结构,所以用一组连续存储单元存放数组的数据元素就有次序约定问题。因此多维数组有行优先顺序和列优先顺序两种存储方式。

3. 稀疏矩阵压缩(三元组法)：在稀疏矩阵中,非零元素的分布是没有规律的,为了进行压缩存储,需要将每一个非零元素的值和其所在的行号、列号作为一个结点存放在一起，这样的结点组成的线性表称作三元组表。因为该结点已不是简单的向量，是无法根据下标进行存取的,所以说稀疏矩阵压缩存储后，必会失去随机存取功能。
4. 从逻辑结构上看，n 维数组由多个 n-1 维的数组构成。
5. 一个长度为 n 的字符串的子串长度为 n(n+1)/2 + 1
6. 在文本编辑程序中查找某一特定单词在文本中出现的位置，可以利用串的**模式匹配**
7. **在串匹配中，一般将主串称为目标串，将子串称为模式串**
8. **数组不能看作是基本线性表的一种推广，因为对数组进行插入可能会导致数组越界**

树和二叉树

1. 二叉树是一种非线性辑结构,但线索二叉树比二叉树多了“线索”这个概念,线索即驱和后继是通过指针去定义的,而指针是 C 语言的一种功能，这就是满足了定义中“使用计算机语言实现逻辑结构”。所以线索二叉树被认为是存储结构,也就是物理结构
2. 对字符进行哈夫曼编码，可以理解成对某个序列构造哈夫曼树
3. 构造哈夫曼树时，最顶层从 0 开始
4. **错题**：一个具有 1025 个结点的二叉树的高度为，11 是错的，(2^10)-1 = 1023， 但是这里说的是二叉树，如果是完全二叉树就是 11，但是二叉树可以只有单边的结点，所以可能有 1025 个，所以答案是 11-1025

图

1. 存储某个图所占存储空间与该图顶点个数相关的是**领接矩阵**
2. n 个顶点的无向连通图，其生成树有 n-1 条边
3. 若带权连通图上各边上的权值互不相同,则该图的最小生成树是唯一的
4. 有向图的顶点有前驱和后继概念，无向图没有
5. 无向图 n 个结点在连通的情况下最少是 n-1 条边
6. 有向图 n 个结点在连通的情况下最少是 n 条边
7. 一般稀疏图用邻接表存储,稠密图用邻接矩阵存储。邻接表只存储非零结点，而邻接矩阵存储所有的结点信息(非零结点与零结点)。稀疏图的非零结点少,所以选用邻接表效率高;稠密图的非零结点多,零结点少,所以选用邻接矩阵。
8. 一个图的邻接矩阵表示是唯一的，但其邻接表是不唯一的。
9. 图的任意两个顶点之间都可能存在联系，因此无法以数据元素在存储区中的物理位置来表示元素之间的关系，即图没有顺序存储结构，但可以借助二维数组来表示元素之间的关系，即**邻接矩阵表示法**。图的链式存储有多种，有邻接表、十字链表和邻接多重表，应根据实际需要的不同选择不同的存储结构。

查找

1. 分块查找要求前一块中的最大关键字必须小于后一块中的最小关键字,且每一块中的关键字不一定有序，即要求表是的**分块有序**。

排序

1. 堆排序本质上是一种树形选择排序。
2. 希尔排序采用分组插入的方法，先将待排序记录分割成几组，从而减少直接插入排序的数据量，然后对每组分别进行直接插入排序，然后增加每组的数据量，重新分组。这样经过几次分组排序后，整个序列中的记录“基本有序”时，再对全体记录进行一次直接插入排序。
3. 希尔排序在一趟排序结束后不一定能选出一个关键字放在其最终位置上。
4. 快速排序是在一次交换过程中就可以消除多个逆序的记录的排序方法。
5. **堆排序所需的时间与待排序的记录个数有关，与待排数据的位置无关**