C语言考试题型/知识点总结

2025.12.30　　娄卫健

一、计算表达式的值（10 分）

核心考察知识点

1. 运算符优先级与结合性和返回值

- 考察范围：逻辑运算符（`!`、`&&`）、算术运算符（`+`、`-`、`*`、`/`）、位运算符（`|`、`<<`）、自增自减运算符（`++`、`--`）(要注意前置自增/减和后置的区别)、赋值运算符（`=`、`-=`）的功能和优先级。
- 关键规则：
  - 优先级：`!`（逻辑非）> 算术运算符 > 位运算符 > 关系运算符 > `&&`（逻辑与）> 赋值运算符。
  - 结合性：除单目运算符（`!`、`++`、`--`）、赋值运算符、条件运算符（不怎么考察结合性）为“右结合” 外，其余为 “左结合”。
- 易错点：
  - 忽略`++b`（前置自增）与`b++`（后置自增）的优先级差异（均高于算术运算符），但执行时机不同（前置先自增再参与运算，后置先参与运算再自增）。
  - 位运算与算术运算混淆（如`x<<2`是左移 2 位，等价于`x*4`，但优先级低于`+`）。
  - 逻辑运算短路特性：`&&`左边为 0 时，右边不再执行；`||`左边为 1 时，右边不再执行（本题虽未直接考察，但属于核心延伸知识点）。
  - **逗号表达式**和**赋值运算符**的返回值。

2. 数据类型与存储特性

- 基本类型：

  short（2 字节）、char（1 字节，区分有符号 / 无符号，默认有符号）、int（2/4 字节，取决于编译器）的取值范围与溢出规则。

  - 示例：`y=0xcd`（十六进制），转换为十进制为 205（有符号`char`中为 - 51，因最高位为 1 表示负数），赋值给`char d`时会发生 “截断”（仅保留低 8 位）。
- 字符串相关：`strlen()`函数的功能（统计字符串中`'\0'`之前的字符个数，不包含`'\0'`），如`"hello"`长度为 5。

3. 结构体、联合体与指针操作

- 结构体数组与指针：`->`（指针访问结构体成员）与`.`（直接访问结构体成员）的区别，如`p->a[1]`等价于`(*p).a[1]`。
- 指针移动：`p++`（后置自增，先使用指针值再移动）、`++p`（前置自增，先移动指针再使用）对结构体成员访问的影响。
- 联合体（共用体）：所有成员共享同一块内存，修改一个成员会影响其他成员，本题中用于解析指令的位字段（`ty:2`表示占 2 个二进制位）。

- 位字段：结构体中指定成员占用的二进制位数（如`struct cmd0 { unsigned char ty:2; unsigned char arg0:6; }`），用于解析紧凑存储的数据（如指令编码）。

4. 常量表示形式

- **整型常量**：
  - 十进制：由 0~9 数字组成，不能以 0 开头（数值 0 除外），例如：`100`、`-50`、`0`。
  - 八进制：以数字 `0` 开头，由 0~7 数字组成，例如：`0123`（对应十进制 83）、`05`（对应十进制 5）。
  - 十六进制：以 `0x` 或 `0X` 开头，由 0~9、a~f（或 A~F）组成，例如：`0x1A`（对应十进制 26）、`0XFF`（对应十进制 255）。

  此外，整型字面常量还可以加后缀指定类型：
  - `u`/`U`：表示无符号整数，例如 `100U`。
  - `l`/`L`：表示长整数，例如 `123L`。
  - `ll`/`LL`：表示长长整数，例如 `456LL`。
- **浮点型字面常量**：表示小数或科学计数法数值，有两种表示形式：
  - 小数形式：必须包含小数点，例如：`3.14`、`0.5`、`-2.0`、`.6`（等价于 0.6）、`5.`（等价于 5.0）。
  - 指数形式：格式为 `尾数e/E指数`，尾数可以是整数或小数，指数必须是整数，例如：`1.23e5`（等价于 1.23×10^5）、`5e-3`（等价于 5×10^{-3}）。

  浮点型字面常量默认是 `double` 类型，加后缀 `f`/`F` 可指定为 `float` 类型（例如 `3.14f`），加 `l`/`L` 可指定为 `long double` 类型（例如 `2.5L`）。
- **字符字面常量**：用单引号 `' '` 括起来的单个字符，分为普通字符和转义字符：
  - 普通字符：例如 `'a'`、`'5'`、`'+'`、`' '`（空格字符）。
  - 转义字符：以反斜杠 `\` 开头的特殊字符，用于表示不可打印或有特殊含义的字符，常见的有：

`\n`	**换行符**
`\t`	水平制表符（Tab）
`\\`	反斜杠本身
`\'`	单引号
`\"`	双引号
`\0`	空字符（ASCII 码值为 0）

- **字符串字面常量**：用双引号 `" "` 括起来的一串字符，C 语言会自动在字符串末尾添加一个 `\0` 作为结束标志。例如：`"hello"`、`"12345"`、`"C language"`、`""`（空字符串，仅包含一个 `\0`）。注意：字符串字面常量和字符字面常量的区别，`'a'` 是字符常量，占 1 字节；`"a"` 是字符串常量，占 2 字节（`'a'` + `\0`）。

需重点学习的内容

- 练习有符号 / 无符号数据的转换（十六进制→十进制→二进制），掌握溢出处理规则。
- 结构体指针与数组的混合操作（`b[2]` 是结构体数组，`p=b` 表示指针指向数组首元素，`p++` 指向 next 结构体）。
- 运算符优先级和返回值的运用。
- strlen、sizeof等的用法和功能
- 位字段的定义与访问（联合体与结构体结合解析紧凑数据的场景）（这个其实考的不多，但是容易忽略）。

例题

```
1. 请根据下面的声明，计算（1）~（5）题表达式的值并填入各题后面的下划线中。
short x=4,y=0xcd,z=0x20;
char s[] = "hello",d;
int a=3,b=4,c=5;
（1）!(a+b)+c-1&&c+b/3 值为：_______
（2）strlen(s) 值为： _______（补充：sizeof(s)值为：_____）
（3）d=y 值为：_______(知识点：赋值运算符,类似还要注意逗号运算符)
（4）c -= a | ++b 值为： _______
（5）x<<2 | y 值为： _______



2. 请根据下面的声明，计算（6）~（10）题表达式的值并填入各题后面的下划线中。
struct T{
int a[3];
char *s;
struct T *x;
}b[2]={{{1,2,3},"HELLO",b},{{-1,0,1},"WORLD",b+1}},*p=b;
（6）++p->a[1] 值为： _______
（7）(*++p).s 值为： _______
（8）(--(b[1].x))->a[2] 值为： _______
（9）(p+1)->a[1]?1:0 值为： _______
（10）(p++)->a[0] + (b[0].x)->a[2] 值为：_______
```

```
1、char stra[ ] = "helloword", strb[15] = { 'h','c','a','f','e','\0' };
int a = 7, b = 4, c = 3;
(1)strlen(stra) 的值为：_______
(2) strlen(strb) 的值为：_______
(3) 表达式(a=7, b=3, a>b?a++:b++,a+b)的值为：_______
(4) 表达式 a += a *= a执行完成后，a的值为：_______
(5) 表达式 !(a-b)+c-2 || b + c/2的值为：_______
```

二、改错题（14 分）（每小问两处错误）

例题

1. 求 2/1、3/2、5/3、8/5、13/8、21/13、… 前 30 项之和。

```
#include <stdio.h>
int main()
{
    int i, t1 = 2, t2 = 1, x;
    float sum = 0;
    for (i = 1; i <= 30; i++) {
        sum += t1 / t2;
        x = t1;
        t1 += t2;
        t2 = x;
     }
    printf("sum = %d\n", sum);
    return 0;
}
```

2. 函数 fun()的功能是计算正整数 num 的各位上的数字平方之和。例如，输入 124，则输出应为21；若输入 326，则输出应为 49。

```
#include<stdio.h>
long fun(long num)
{
    long k = 1;
    do{
        k+= (num%10)*(num%10);
        num/= 10;
    } while(num);
}
int main()
{
    long n;
    printf(“ Please enter a number: ”);
    scanf(“ %ld ”, &n);
    printf(“ %ld\n ”, fun(n));
    return 0;
}
```

3. 函数 fun( )的功能是将字符串 s 中位于偶数位置（最左边为第 0 位）的字符或 ASCCII 码为奇数的字符放入字符串 t 中。例如，若 s 为 ADFESH，则 t 应是 AFES。

```
void fun(char s[ ], char t[ ])
{
    int i, j = 0;
    for(i = 0; i<strlen(s); i++)
    if(i%2 = 0 || s[i]%2 != 0)
    t[++j] = s[i];
    t[j] = '\0';
}
```

4. 逆序输出数组元素。

```
#include<stdio.h>
#define N 6
int main()
{
    int i,a[N]={2,5,7,14,19,22};
    for(i=N;i>=0;i--)
    printf("%d ",(a+i));
    return 0;
}
```

5. 函数 fun 的功能是删除字符串 s 中第一次出现的字符'b'，函数返回一个指向最终的字符串 s 的指针，若字符串 s 中不存在字符'b'，则返回 NULL。例如，输入 qwebrtb，则输出应为 qwertb；若输入 qwer，则输出应为 Not exist!。

```
#include <stdio.h>
char *fun(char s[ ] )
{
    int i,k=0;
    while(s[k] != '\0') {
        if( s[k] ='b') continue;
        k++;
    }
    if(s[k] == '\0') return s;
    else {
        do {
            s[k]=s[k+1];
            k++;
        } while(s[k] != '\0');
        return s;
    }
}
int main()
{
    char s[30], *p;
    fgets(s,30,stdin);
    p=fun(s);
    if(p==NULL) printf("Not exist!\n");
    else printf("%s\n",p);
return 0;
```

```
}
```

6. 函数 fun()的功能是：在字符串 s 中找出 ASCII 码最小的字符，将其放在最左边位置上，并将该字符前的原字符向后顺序移动。例如，调用 fun()函数之前字符串为 ebfAgCDg，调用后字符串为 AebfgCDg。

```
void fun(char s[ ] )
{
    char min, *q;
    int i = 0;
    min = s[i];
    while(s[i] != 0) {
        if(min>s[i]) {
            q = &(s+i);
            min = s[i];
        }
        i++;
    }
    while(q>s) {
        *q = *q-1;
        q--;
    }
    s[0] = min;
}
```

7. 函数 fun()的功能是：从 n 个学生的成绩中统计出高于平均分的学生人数，人数由函数值返回，平均分存放在形参 average 所指的存储单元中。假如 6 名学生的成绩为:84 65.5 68 96.5 87 56，则高于平均分的学生人数为 3 (平均分为 76.1666667)。

```
int fun(float *s, int n, float *average)
{
    float ave, t = 0;
    int count = 0, k, i;
    for(k = 0; k<n; k++) t+=s[k];
    ave = t/n;
    for(i = 0; i<n; i++)
        if(s[i]>ave) count+1;
    average = ave;
    return count;
}
```

核心考察知识点（每小题 2 个错误，覆盖语法、逻辑、库函数使用）

1. 语法错误

- 输入输出格式符：`printf("sum = %dn", sum);` 中`%d`与`float`类型不匹配（应改为`%f`）；`scanf(* %ld ", &n);` 中格式字符串多余`*`（应改为`"%ld"`）。
- 函数返回值：有返回值的函数（如`long fun(long num)`）必须有`return`语句，否则返回随机值。

- 运算符错误：`if(i%2 = 0)` 中 `=` 是赋值运算符，判断相等应使用 `==`；`count+1` 是表达式，未赋值给 `count`（应改为 `count++` 或 `count+=1`）。
- 数组访问：数组下标从 0 开始，逆序输出时 `for(i=N;i>=0;i--)` 会访问 `a[N]`（越界，应改为 `i=N-1`）。
- 指针与数组：`printf("%d ",(a+i));` 中 `a+i` 是指针地址，应解引用为 `*(a+i)` 或 `a[i]`。

2. 逻辑错误

- 整数除法：`sum +=t1/t2;` 中 `t1` 和 `t2` 均为 `int`，结果为整数除法（如 `2/1=2`，`3/2=1`），应改为 `sum += (float)t1/t2;`（强制类型转换，确保浮点数除法）。
- 变量初始化：`long k=1;` 用于累加数字平方和，初始值应为 0（否则会多算 1，如输入 124 时结果为 22 而非 21）。
- 字符串拼接：`t[j] = '\0'; t[++j] = s[i];` 逻辑颠倒，应先赋值 `t[j] = s[i];`，再 `j++`，最后在循环结束后添加 `t[j] = '\0'`（否则字符串无结束符）。
- 字符删除逻辑：`if( s[k] =='b') continue;` 仅跳过 `'b'`，未实现删除（应将后续字符前移覆盖 `'b'`，即 `s[k] = s[k+1]`）；返回值逻辑错误（`if(s[k] == '\0') return s;` 应为 “未找到 `'b'` 返回 NULL，找到后删除并返回 `s`”）。
- 指针操作：`q = &(s+i);` 语法错误，`s+i` 是指针，取地址应为 `q = s + i`（或 `&s[i]`）；`*q = *q-1;` 应改为 `*q = *(q-1)`（将前一个字符后移）。
- 形参赋值：`average = ave;` 中 `average` 是指针，应改为 `*average = ave;`（将平均分存入指针指向的内存单元，否则主函数无法获取结果）。
- 宏定义是简单的替换，需要加括号来限制运算顺序，否则可能出错，该特性后面也会考到。

3. 库函数使用

- 字符串函数：`strlen()` 需包含头文件 `<string.h>`，否则编译警告。
- 文件操作函数：`fgets()` 读取字符串时会包含换行符 `'\n'`，`fopen()` 打开文件后必须 `fclose()`，否则内存泄漏。
- 内存分配函数：`malloc()` 返回 `void*`，需强制转换为目标指针类型（如 `struct intnode *p = (struct intnode *)malloc(sizeof(struct intnode))`），且需检查分配是否成功（避免 NULL 指针操作）。

需重点学习的内容

- 语法细节：分号、括号、格式符(int,long long int ,float,double,char,char[]对应格式符)的正确性，强制类型转换的使用场景。
- 逻辑梳理：明确程序功能（如求和、删除字符、统计数字平方和）。
- 库函数：牢记常用函数的参数、返回值、头文件（如 `strlen()`、`strstr()`、`malloc()`、`fopen()`）。
- 指针与数组：区分指针地址与指针指向的值（`a+i` 是地址，`*(a+i)` 是值）。
- switch的详细用法，尤其是没有break的话会自动进入下一个判断（后面也有涉及）。
- 函数传参时用值传递，注意值传递的特性。

三、简答题（24 分）

例题

1.一个 2 字节的 short 型变量的最大允许值是多少？如果再加 1，结果是多少？

2.请写出一个 C 表达式，如果字符变量 ch 的值是大写字母，则将其值修改成小写字母；如果字符变量 ch 的值是小写字母，则将其值修改成大写字母;如果 ch 的值不是字母，其值不变。
//考C语言最好记住大小写字母、数字对应的ASCII码值。

3.请写一个 C 表达式，将 int 型变量 x 的最高二进制位设置为 1，其余位不变，要求所写表达式在 int 为 2 或 4 字节的机器上均能正确执行。

4. 下列程序段定义了 3 个变量 i，类型分别为 int、long 与 float。指出每个变量的存储类，它们分别在那些行进行了声明和使用。

```
1 int i;
2 void fun(long i)
3 {
4   long x=i;
5   {
6       float i;
7       i=3.4;
8   }
9   x=i+2;
10 }
11 int *p=&i;
```

5. 判断以下叙述是否正确，如果不正确，请说明原因。
(1) 表达式 z+=x+++y 中的词法元素（记号）数目是 7 个。
(2) 若有定义 char x[ ]="abcde", y[ ]={'a','b','c','d','e'}; 则数组 x 和数组 y 等价。
(3) typedef char *STRING 和 #define STRING char* 中命名的 STRING 使用上没有区别。
(4) 变量可以命名为 INT。

6. 写出下列声明语句。
（1） f 是值为 3 的“只读”型整型变量;

（2） g 是一个无参指针函数，其返回值是指向包含 2 个指针元素的数组的指针，该数组中每个元素是指向有一个字符指针参数且无返回值的函数。

3. 请写一个C表达式，将int型数x向左循环移动n位，循环左移即把移出的高位放到该数的低位，n值小于int型数所占的位数。

4. 请定义一个标准宏MIN(x,y,z)，这个宏返回三个参数中最小的一个。

5. 请说明下面声明中p1和p2的含义与区别。

```
int x=2;
const  int *p1 =&x;
int *const p2 =&x;
```

6. 请根据下面各题的解释，写出对应的声明语句。
(1)a是一个指向有10个整型数数组的指针。Int (*a)[10]
(2) b是一个有10个元素的指针数组，数组元素是指向函数的指针，该函数有一个整型参数并返回一个整型数。Int (*b[10])(int)

核心考察知识点

1. 数据类型取值范围与溢出

2. 字符大小写转换与条件表达式

3. 位运算（设置最高位为 1、循环位移等）

4. 存储类与变量声明 / 使用

- 全局变量：存储在静态区，默认初始化为 0，作用域为整个文件，生命周期为程序运行期间。
- 函数参数：存储在栈区，作用域为函数内部，生命周期为函数调用期间，属于自动变量。
- 局部变量：存储在栈区，作用域为函数内部，同名局部变量屏蔽外部变量。

5. 语法与语义判断

- (1) 错误：表达式`z+=x+++y`的词法分析遵循 “最长匹配原则”，解析为`z += (x++) + y`，词法元素为：`z`、`+=`、`x`、`++`、`+`、`y`，共 6 个，而非 7 个。
- (2) 错误：数组`x[]="abcde"`隐式包含`'\0'`（长度为 6），数组`y[]={'a','b','c','d','e'}`无`'\0'`（长度为 5），因此`x`是字符串（可被`strlen()`等函数处理），`y`是字符数组（不是字符串，使用字符串函数会越界）。
- (3) 错误：typedef char* STRING是类型定义（定义STRING为char*类型），#define STRING char*是宏替换（简单文本替换）。
  - 差异示例：`STRING s1, s2;`，`typedef`定义中`s1`和`s2`均为`char*`；`#define`替换后为`char* s1, s2;`，其中`s2`是`char`型（仅`s1`是指针）。
- (4) 正确：`INT`是合法标识符（C 语言标识符区分大小写，`int`是关键字，但`INT`不是），可作为变量名。

6. 复杂声明语句

- (1) 只读整型变量（值为 3）：`const int f = 3;`（`const`修饰变量为只读，不可修改；若需 “真常量”，可使用`#define f 3`，但二者本质不同：`const`是变量，有类型，`#define`是宏，无类型）。
- (2) 无参指针函数，返回值是 “指向包含 2 个指针元素的数组的指针”，数组元素是 “指向有一个字符指针参数且无返回值的函数”：

- 声明：`void (*(*g())[2])(char*);`
- 解析：
  - `g()`：`g`是无参函数。
  - `*g()`：`g`的返回值是指针。
  - `(*g())[2]`：该指针指向一个包含 2 个元素的数组。
  - `*(*g())[2]`：数组的每个元素是指针。
  - `void (*(*g())[2])(char*)`：该指针指向一个无返回值、参数为`char*`的函数。

需重点学习的内容

- 数据类型取值范围与溢出规则（尤其是有符号 / 无符号的区别）。
- 位运算的灵活应用（设置位、清除位、翻转位、提取位）。
- 存储类（`static`、`extern`、`register`）的作用域、生命周期、存储位置。
- 复杂声明的解析方法（从内向外，先看函数 / 数组，再看指针 / 类型）。
- `typedef`与`#define`的区别（类型定义 vs 文本替换）。

四、程序分析题（24 分）

例题

```
1. 写出下面程序的运行结果。
#include<stdio.h>
long fib(int g)
{
    switch (g) {
    case 0: return 0;
    case 1: case 2: return 1;
    }
    return(fib(g-1)+fib(g-2));
}
int main()
{
    long k;
    k=fib(7);
    printf("k=%d\n",k);
    return 0;
}
//k=13


2. 写出下面程序的运行结果。
#include <stdio.h>
int main()
{
    int i,r;
    char s1[80]="bus",s2[80]="book";
    for (i=r=0;s1[i]!='\0'&&s2[i]!='\0';i++)
        if (s1[i]==s2[i]) i++;//注意这里也有i++
        else { r=s1[i]-s2[i]; break; }
    printf("%d",r);
```

```
    return 0;
}
//4
```

3. 写出下面程序的运行结果。

```
#include <stdio.h>
#define M 5
#define f(x) x*x
#define ff(x) (x*x)
int main()
{
    int n1,n2;
    n1=100/f(M);
    n2=100/ff(M);
    printf("n1=%d,n2=%d\n",n1,n2);
    return 0;
}
//n1=100,n2=4
```

4. 写出下面程序的运行结果。

```
#include <stdio.h>
#define M 4
void fun(int *a)
{
    int i,j,k,m;
    for(i=M;i>0;i--) {
        k=*(a+M-1);
        for (j=M-2;j>=0;j--)
            *(a+j+1)=*(a+j);
        *a=k;
        for(m=0;m<M;m++)
            printf("%d",*(a+m));
        printf("\n");
    }
}
int main()
{
    int a[M]={1,2,3,4};
    fun(a);
    return 0;
}
/*
4123
3412
2341
1234
*/
```

5. 写出下面程序的运行结果。

```
#include <stdio.h>
void amov(int *p, int (*a)[3], int n)
{
    int i,j,s=0;
```

```
    for (i=0;i<n;i++)
        for(j=0;j<n;j++) {
            *p=(*(a+j))[i];
            s=s+*p;
            p++;
        }
    printf("\ns=%d\n",s);
}
int main()
{
    int i,*p,a[3][3]={{1,3,5},{2,4,6},{7,8,9}};
    p=&a[0][0];
    for (i=0;i<3;i++) printf("%d ",*(p+i*3));
    putchar('\n');
    for (i=0;i<3;i++) printf("%d ",*(*(a+i)+i));
    amov(p,a,3);
    return 0;
}
*/
 1  2  7        （1分）
 1  4  9        （1分）
 S=48           （2分）
*/
```

核心考察知识点（实际上根据代码一步一步细心模拟即可，一定要细心，这里给的知识点只是针对这套卷）

1. 递归函数（斐波那契数列）

- 递归逻辑：`fib(g)` 的定义为 `g=0→0`，`g=1或2→1`，`g>2→fib(g-1)+fib(g-2)`。
- 计算过程：`fib(7) = fib(6)+fib(5) = (fib(5)+fib(4))+(fib(4)+fib(3)) = ... = 13`。
- 考点：递归的终止条件、递归调用栈的过程（延伸：递归的效率问题，可通过迭代优化）。

2. 字符串比较与循环逻辑

- 考点：循环条件、字符串字符的 ASCII 码比较、`continue` 与 `break` 的区别。

3. 宏定义（带参数宏的陷阱）

- 宏替换规则：带参数宏是 “文本替换”，无类型检查，不计算表达式值。
  - `#define f(x) x*x`：`f(M)` 替换为 `M*M`，`100/f(M)` 即 `100/5*5`（优先级：`/` 和 `*` 相同，左结合），结果为 `100/5*5=20*5=100`。
  - `#define ff(x) (x*x)`：`ff(M)` 替换为 `(5*5)`，`100/ff(M)` 即 `100/(5*5)=100/25=4`。
- 运行结果：`n1=100,n2=4`。
- 考点：带参数宏的替换陷阱（缺少括号导致优先级错误），宏与函数的区别（宏无调用开销，但可能产生副作用，如 `f(x++)` 会替换为 `x++*x++`，执行两次自增）。

4. 数组移位（右移操作）

- 程序逻辑：函数fun将数组a

  的元素循环右移，每次移 1 位，共移M次（M=4）。

  - 初始数组：`[1,2,3,4]`。
  - 第 1 次右移：保存最后一个元素`4`，其余元素后移，结果`[4,1,2,3]`。
  - 第 2 次右移：保存`3`，结果`[3,4,1,2]`。
  - 第 3 次右移：保存`2`，结果`[2,3,4,1]`。
  - 第 4 次右移：保存`1`，结果`[1,2,3,4]`。
- 运行结果（每行输出一次移位后的数组）

```
4123
3412
2341
1234
```

- 考点：数组元素的移动（通过指针访问数组元素`*(a+j)`），循环嵌套的逻辑（外层控制移位次数，内层控制元素后移）。

5. 矩阵转置与求和

- 程序逻辑：
  - 第一个循环：输出数组`a[3][3]`的第 0 列元素（`a[0][0]`、`a[1][0]`、`a[2][0]`），即`1 2 7`。
  - 第二个循环：输出矩阵主对角线元素（`a[0][0]`、`a[1][1]`、`a[2][2]`），即`1 4 9`。
  - 函数`amov`：将矩阵`a`转置（`(*(a+j))[i]`即`a[j][i]`，赋值给`*p`，即`a[i][j]`），并累加所有元素和（`1+3+5+2+4+6+7+8+9=45`）。
- 运行结果：

```
1 2 7
1 4 9
s=45
```

- 考点：二维数组的指针访问（`*(a+i)`是第`i`行的首地址，`*(*(a+i)+j)`是`a[i][j]`），矩阵转置的逻辑（行列互换），累加求和。（**二维数组在内存中的存储**）

6. 文件操作与命令行参数（该题我未整理）

- 命令行参数：`argc`是参数个数（包含程序名），`argv[]`是参数数组（`argv[0]`是程序名，`argv[1]`~`argv[argc-1]`是传入的文件名）。
- 程序逻辑：输入e12_42 f1 f2 f3，argc=4，argv[1]="f1"、argv[2]="f2"、argv[3]="f3"。
  - 函数`sub`：读取文件中`'!'`之前的字符，每个字符加 1（`a→b`、`b→c`、`c→d`）。
  - 文件`f1`内容`aaa!`→输出`bbb`；`f2`→`bbb!`→`ccc`；`f3`→`ccc!`→`ddd`。
- 运行结果：`bbbcccddd`。
- 考点：文件打开（`fopen`）、读取（`getc`）、关闭（`fclose`），命令行参数的使用，字符 ASCII 码的修改（`c+1`）。

需重点学习的内容

- 最重要：根据题目一点一点去模拟（如果能知道函数作用也可以）
- 递归函数的逻辑分析（画递归调用树）。
- 带参数宏的替换规则与陷阱（务必加括号避免优先级问题）。
- 数组与指针的等价性（`a[i]`等价于`*(a+i)`，二维数组`a[i][j]`等价于`*(*(a+i)+j)`）。
- 文件操作的基本流程（打开→读写→关闭），命令行参数的使用场景。
- 矩阵转置、数组移位等常见算法的实现逻辑。

五、程序完善题（28 分）

例题

1. 函数 selSort(int a[], int n)采用选择排序法，对长度为 n 的一维整型数组 a 进行降序排序。

```
void selSort(int a[], int n)
{
    int i, j;
    for (i=0; i<n-1; i++) {
        int idx = i;
        for (j=i+1; j<n; j++) {
            if ( ①_____ < a[j])
                idx = j;
        }
        if (idx != i)
            a[i] = a[i] + a[idx], a[idx] = ②_____ , a[i] = ③_____ ;
    }
}
```

2. 函数 delSubstr(char *str, const char *substr)用于删除母串 str 中所出现的所有子串 substr。该函数在实现时用到了标准函数库 string.h 中的字符串处理函数。

```
#include <string.h>
void delSubstr (char *str, const char *substr)
{
    int lenm = strlen(str), lens = strlen(substr);
    char *pch = str;
    int i, j;
    while ((pch=strstr( ①_____ , substr)) != NULL) {
        for (i=0; i<lens; i++) *pch++ = '\0';
    }
    for (pch=str,i=0; i<lenm; i++) {
        if (*(str+i) != '\0')
            *pch++ = ②_____ ;
    }
    *pch = ③_____;
}
```

4. 下面是十进制超大整数加法运算程序中的几个函数，请完善代码。

```
#include <stdio.h>
#include <stdlib.h>
struct intnode {
    int data;
    struct intnode *next;
};
/* 采用后进先出方式建立链表，存放输入的十进制超大整数 */
void inputdata( ①_____ phd)
{
    char ch;
    struct intnode *p;
    while ((ch=getchar()) != '\n') {
        p = (struct intnode *)malloc(sizeof(struct intnode));
        p->data = ch - '0';
        p->next = ②_____ ;
        *phd = p;
    }
}
/* 超大整数的加法运算，采用先进先出方式创建和值链表 */
struct intnode *exadder(struct intnode *p1, struct intnode *p2)
{
    struct intnode *head, *tail, *p;
    int carry = 0;
    head = tail = (struct intnode *)malloc(sizeof(struct intnode));
    while (p1!=NULL && p2!=NULL) {
        tail->next = (struct intnode *)malloc(sizeof(struct intnode));
        tail = ③_____;
        carry += p1->data + p2->data;
        tail->data = carry % 10 + '0';
        carry /= 10;
        p1 = p1->next, p2 = p2->next;
    }
    p = p1!=NULL ? ④_____ ;
    while (p != NULL) {
        tail->next = (struct intnode *)malloc(sizeof(struct intnode));
        tail = tail->next;
        carry += p->data;
        tail->data = carry % 10 + '0';
        carry /= 10;
        p = p->next;
    }
    if (carry) {
        tail->next = (struct intnode *)malloc(sizeof(struct intnode));
        tail = tail->next;
        tail->data = carry + '0';
    }
    tail->next = ⑤_____ ;
    p = head->next;
    free(head);
    return p;
}
```

核心考察知识点

题目风格与C语言实验的题目有些类似，这道题对大家的代码能力有一定考察，也是最没有套路的题目，只要平常有写代码，做几道题就知道怎么写了，下面给出的知识点也是仅仅只能针对这道题，但是逻辑都是类似的，重点关注链表操作（如插入、反转、删除等）

1. 选择排序（降序）

- 算法逻辑：每次遍历找到未排序部分的最大值，与未排序部分的第一个元素交换。
- 填空解析：
  - ① `a[idx]`：比较未排序部分的当前最大值（`a[idx]`）与当前元素（`a[j]`），若`a[idx] < a[j]`，则更新最大值索引`idx`。
  - ② `a[i] - a[idx]`：交换`a[i]`与`a[idx]`（利用加减法，无需临时变量），步骤为`a[i] = a[i] + a[idx]; a[idx] = a[i] - a[idx]; a[i] = a[i] - a[idx];`。
  - ③ `a[i] - a[idx]`：完成交换的最后一步（最终`a[i]`存储原`a[idx]`的值，`a[idx]`存储原`a[i]`的值）。
- 考点：选择排序算法的原理，元素交换的实现方式（临时变量法、加减法、异或法）。

2. 字符串删除子串

- 算法逻辑：先找到子串`substr`在`str`中的位置，将子串替换为`'\0'`（标记删除），再将剩余字符拼接成新字符串。
- 填空解析：
  - ① `str`：`strstr()`函数的第一个参数是母串，需从`str`开始查找（每次查找后`pch`指向子串起始位置，下次查找应从`pch + lens`开始，原程序逻辑需优化，但按题目填空要求填`str`）。
  - ② `*(str + i)`：将母串中非`'\0'`的字符复制到`pch`指向的位置（即保留未被删除的字符）。
  - ③ `'\0'`：在新字符串末尾添加结束符。
- 考点：`strstr()`函数的使用（查找子串位置），字符串的拼接与结束符的添加。

3. 指令解析（联合体与函数指针数组）

- 算法逻辑：通过联合体解析指令的类型码，提取参数，调用对应的功能函数。
- 填空解析：
  - ① `(inst >> 6) & 0x3`：指令的最高 2 位是类型码，`inst >> 6`将最高 2 位移到最低位，`&0x3`（二进制`11`）提取低 2 位，得到类型码（0~3）。
  - ② `oneins.ins0.arg0`：`cmd0`类型的指令，参数`arg0`占低 6 位，直接通过联合体成员访问。
  - ③ `exe[typ]()`：函数指针数组`exe`存储 4 个功能函数的地址，`exe[typ]`获取对应类型的函数指针，`()`调用该函数并返回结果。
- 考点：联合体解析位字段，函数指针数组的定义与使用（`int (*exe[])(void) = {func0, func1, func2, func3};`），位运算提取指定字段。

4. 超大整数加法（链表操作）

- 算法逻辑：用链表存储超大整数（每个节点存 1 位数字，后进先出建立链表，即低位在前、高位在后），加法时按位相加，处理进位，结果链表先进先出（高位在前、低位在后）。
- 填空解析：
  - ① `struct intnode **`：`inputdata` 函数需修改头指针（`phd` 是头指针的指针），因链表建立时头指针会不断更新（每次新节点成为新头）。
  - ② `*phd`：新节点的 `next` 指向当前头指针指向的节点（`*phd`），实现后进先出（新节点插入表头）。
  - ③ `tail->next`：创建新节点后，`tail` 移动到新节点（保持 `tail` 始终指向结果链表的末尾）。
  - ④ `p1 : p2`：三元运算符，当 `p1` 不为空时取 `p1`，否则取 `p2`（处理两个数长度不一致的情况，继续累加剩余 digits）。
  - ⑤ `NULL`：结果链表的末尾添加 `NULL`，标记链表结束。
- 考点：链表的建立（头插法）、遍历、节点插入，超大整数加法的进位处理，三元运算符的使用。

需重点学习的内容

- 经典排序算法：选择排序、冒泡排序、插入排序的原理与实现（降序 / 升序切换）。
- 字符串处理：`strstr()`、`strlen()` 等函数的组合使用，子串删除、替换的常见逻辑。
- 函数指针与指针数组：函数指针的定义、函数指针数组的应用（如回调函数、指令分发）。
- 链表操作：头插法、尾插法建立链表，链表的遍历与节点操作，超大数运算（加法、减法）的进位 / 借位处理。
- 位运算与联合体：紧凑数据的解析（如指令、协议帧），位字段的定义与访问。

总结：整体考察重点与学习方向

1. 核心基础（占比约 40%）

- 运算符优先级与结合性、数据类型与存储特性、基本语法（分号、括号、格式符）。
- 字符串与数组操作、简单指针应用（数组指针、结构体指针）。
- 库函数使用（`strlen()`、`strstr()`、`malloc()`、`fopen()` 等）。

2. 进阶应用（占比约 35%）

- 复杂指针操作（函数指针、指针数组、二维数组指针）。
- 结构体、联合体（位字段）的应用。
- 递归函数、排序算法、数组移位、矩阵转置等基础算法。
- 文件操作与命令行参数。

3. 易错点与陷阱（占比约 25%）

- 宏定义的文本替换陷阱（优先级、副作用）。
- 有符号 / 无符号数据的转换与溢出。
- 指针地址与指针指向的值的混淆。
- 字符串结束符 `'\0'` 的遗漏。

- 函数返回值缺失、类型不匹配。