2023-2024秋季学期
程序设计作业解析

第一次作业解析

1.

	9	64	85	-53	-111	2499	-28654	21019
八进制	011	0100	0125	0313(8bits)/0177713(16bits)	0221(8bits)/0177621(16bits)	04703	0110022	51033(051033)
十六进制	0x9	0x40	0x55	0xffcb/0xffff ffcb (更正 0xcb/0xffcb)	0xff91/0xffff ff91 (更正 0x91/0xff91))	0x9c3	0x9012/0xffff9012	0x521b

注：采用补码的形式来表示，对于负数，要确保没有发生溢出，则要根据数的大小决定采用8bits、16bits 还是 32bits 来表示数字。这道题没有规定数字的大小，所以具体位数可以自己决定。

2.

习题 1-2. B 表示二进制，O 表示八进制，H 表示十六进制，D 表示十进制

(100101101)B =301D

(67.7225)D = 1000011.1011(1000111101010111000010)B（括号括起来的是循环节，当进行进制转换时，有限小数可能会变成无限循环小数）

(2B6)H = 694D

(4702.504)O = 100111000010.1010001B

(2B.9E)H = 101011.1001 1110 0B = 53.474O

3.

习题 1-3. 仿照教材例 1.3 写出输出 1 到 20 平方的程序

```
#include<stdio.h>
int main(void)
{
	int i=1;
	for(;i<=20;i++){
		printf("%d*%d=%d\n",i,i,i*i);
	}
	return 0;
}
```

第二次作业解析

- 习题 1-3：教材 P39 T2

① E-4，科学计数法，缺少尾数，不合语法
（科学计数法表示的数字需要同时具有尾数和指数）
② A423，A 不是十进制数
③ -1E-31，科学计数法，浮点数
④ 0xABCL，十六进制 long int 型
⑤ .32E31，科学计数法，浮点数
⑥ 087，“8”不是八进制数的数字
⑦ 0xL，没有表示任何具体数字
⑧ 003，八进制数
⑨ 0x12.5，c 中没有十六进制小数表示方法
⑩ 077，八进制数
⑪ 11E，科学计数法，缺少指数，不合语法
⑫ 056L，八进制 long int 型
⑬ 0.，有小数点，是浮点数（小数部分默认为 0）
⑭ .0，有小数点，是浮点数（整数部分默认为 0）

- 整型常量：（4）（8）（10）（12）
- 实型常量：（3）（5）（13）（14）

2.

5. 写出以下程序的输出结果。

```
#include<stdio.h>
int main()
{  int a=8,b=9;float x=127.895,y=-123.456;
   char c='B';long n=12345678L;unsigned u=65535u;
   printf("%d,%d\n",a,b);
   printf("%5d,%5d\n",a,b);
   printf("%f,%f\n",x,y);
   printf("%-12f,%-12f\n",x,y);
   printf("%8.3f,%8.3f,%.3f,%.3f,%4f,%5f\n",x,y,x,y,x,y);
   printf("%e,%10.4e\n",x,y);
   printf("%c,%d,%o,%x\n",c,c,c,c);
   printf("%ld,%lo,%lx\n",n,n,n);
   printf("%u,%o,%x,%d\n",u,u,u,u);
   printf("%s,%6.3s,%-10.5s\n","C language","C language","C language");
   return 0;
}
```

8,9

1. □□□□8,□□□□9（□代表空格）

127.895000,-123.456000(127.894997,-123.456001)（由于浮点数采用类似科学计数法的存储方式，无法精确表示数字，因此可能和期望存储的值存在一定误差，以计算机实际输出为准）

127.895000□□,-123.456000□(127.894997□□,-123.456001□)

□127.895,-123.456,127.895,-123.456,127.895000,-123.456000(127.894997,-123.456001)

1.278950e+002,-1.2346e+002(1.278950e+02,-1.2346e+02)（小数点、加号、e、指数等都会计入字符宽度，实际输出宽度大于最小输出长度限制，因此最小输出长度的占位符修饰不起作用；另外，如果编译器不同，指数输出的长度可能会不一样，以实际输出为准）

B,66,102,42

12345678,57060516,bc614e

65535,177777,ffff,65535（思考：如果是 unsigned short u = 65535u; printf(“%hhd”,u); 结果会如何？答案-1）
C□language,□□□C□l, C□lan □□□□□

6. 试按以下变量定义，分别用函数 scanf()和 cin()流编写为它们输入值的代码。

```
int i;char c;long k;float f;double x;
```

```
#include<stdio.h>
int main()
{
	int i;
	char c;
	long k;
	float f;
	double x;
	scanf(“%d%c%ld%f%lf”,&i,&c,&k,&f,&x);
	return 0;
}
```

```
#include<iostream>
using namespace std;
int main()
{
	int i;
	char c;
	long k;
	float f;
	double x;
	cin >> i >>c>>k>>f>>x;
	return 0;
}
```

注：cin 是 c++ std 命名空间中的标准输入流，在 iostream 头文件中，大家只作了解即可，scanf 才是需要熟练掌握的函数

第三次作业解析

教材 P40 T10

10. 试分别用最紧凑的一条 C 代码描述完成下列要求的计算。
（1）把整型变量 i 和 j 的和赋给整型变量 k，并同时让 i 的值增加 1。
（2）把整型变量 x 值扩大一倍。
（3）在变量 i 减去 1 后，将变量 j 减去变量 i。
（4）计算变量 i 除以变量 j 的余数 r。
（5）将实型变量 x 精确到小数点后第三位四舍五入后的值赋给实型变量 y。

(1) k=(i++)+j;（建议对计算顺序拿不准时把括号加上，不加括号可能会错，加了括号一定不会出错）

(2) x<<=1;或者 x*=2;

<<是位移操作符，可以搜一下做了解，不要求大家一定掌握

(3) j-=--i;

(4) r=i%j;

(5) y =(int)(x*1000.0+fabs(x)/x*0.5)/1000.0; 或 y = round(x * 1000.0) / 1000.0;

本题思路是先将变量扩大 1000 倍然后根据正负性加减 0.5 再取整，然后再缩小 1000 倍。绝大多数同学只考虑了正数情况，或者四舍五入的位数选错了，还有的同学是用格式化符%.3f 来保留小数点后位数的，但是这样不能实现四舍五入的功能。另外，round 函数可以返回值的四舍五入整数值（需要 include math.h）

教材 P40 T11

11. 试用 C 语言表达式描述以下数学计算式或逻辑条件。
（1）$V=(4/3)\pi r^3$;
（2）R=1/(1/R1+1/R2);
（3）$y=x^5+x^3+6$;
（4）F=GM1M2/R2;
（5）sin(x)/x + |cos(πx/2)|;
（6）0<a<10;
（7）条件 x=1 与 y=2 有且只有一个成立。

1. V=4.0/3*M_PI *r * r * r;（需要使用 math.h 头文件）

注意 4 和 3 都是 int 型常数，表达式 4/3 会得到 int 型的结果 1，所以要用 4.0/3 或者(double)4/3 将其转为 double 型

M_PI 是 math.h 中定义的 pi 值，include math.h 后可以直接使用 M_PI 来代替 pi 的值

希腊字母在 c 中不合法，因此不能用Π和π这样的符号

c 中的幂不能用^这个符号，替代方法为 math.h 中定义的 pow()函数

2.R=1/(1/(double)R1+1/(double)R2);或 R=1.0/(1.0/R1+1.0/R2);

将 R1 和 R2 转为 double 型，避免 R1 和 R2 为整型时被 1 除得到 int 型的结果

只要保证除式两边的变量有一个是 double 型，得到的结果即为 double 型

3.y=x * x * x * x * x + x*x *x +6;或 y=pow(x,5)+pow(x,3)+6;（需要使用 math.h 头文件）

4.F=G * M1* M2/((double) R * R);

万有引力公式，书上的题目表达的不是很清楚，分母的 R*R 两边需要加括号，否则计算顺序不对

5.sin(x)/x + fabs(cos(M_PI*x/2))

int 型的绝对值可以用 abs()函数计算,浮点型的绝对值则要使用 fabs()函数

6.(a>0) && (a<10)

如果对运算符的优先级不太熟悉，则建议逻辑表达式多加些括号避免歧义

7.(x==1&&y!=2)||(x!=1&&y==2)或(x==1)!=(y==2)或(x==1)^(y==2)

^为异或运算符，若 c=a^b，则当且仅当 a 和 b 中有且仅有一个为真、另一个为假时，c 为真

12. 设在求以下表达式之前，整型变量 a 的值是 4，试指出在求了以下表达式之后，变量 a、b 和 c 的值。

（1）b=a*a++;

（2）c=++a + a;

1. a=5,b=20;

这道题很多同学的答案是 b=16，按理来说确实是这样，但是实际运行的结果是 20，这是为什么呢?

我看了一下编译后的汇编代码发现：程序首先先将 a=4 放入了临时变量 temp1，然后对 a 进行了+1 操作，然后将 a=5 放入了临时变量 temp2，最后用 temp1 * temp2 得到了 b 的值，即 b=20

一种可能的解释是：由于++运算的优先级最高，所以计算机先对 a++进行了处理，即取到 a 的值 4 后再对它+1 变成 5，然后再处理 a*a++时把自增后的值赋给了第一个 a，于是表达式就变成了 5*4=20。

确实有些古怪…所以这道题的答案为 b=16 的同学也不会扣分

这道题也许是想让大家明白——以后自己写程序时千万不要使用这种带歧义的表达式!

2.a=5,c=10;

教材 P40 T15

15. 编写输入 3 个整数，输出这 3 个数的和、平均值、最小值和最大值的程序。

```
int a =1, b=2, c=3;
int max, min;
int sum = a + b +c;
float average = (a+b+c)/3.0; //平均值有可能是小数，所以要定义为 float，并且是除以 3.0 而不是除以 3，否则会得到取整后的结果
if(b>a){
        max = b;
        min = a;
}
else{
        min = b;
        max = a;
}
if(c>max)
```

```
        max = c;
else if (c < min)
        min = c;
```

第四次作业解析

1. 编写一个程序，输入一个整数，输出 0～9 各个数字在该整数中出现的次数。

```c
#include <stdio.h>

int main()
{
   int n0, n1, n2, n3, n4, n5, n6, n7, n8, n9;
   n0 = n1 = n2 = n3 = n4 = n5 = n6 = n7 = n8 = n9 = 0;
   int x;
   int t;

   scanf("%d", &x);

   while(x)
   {
     t = x % 10;
     switch(t)
     {
     case 0: n0++; break;
     case 1: n1++; break;
     case 2: n2++; break;
     case 3: n3++; break;
     case 4: n4++; break;
     case 5: n5++; break;
     case 6: n6++; break;
     case 7: n7++; break;
     case 8: n8++; break;
     case 9: n9++; break;
     }

     x /= 10;
   }

   printf("n0 = %d, n1 = %d, n2 = %d, n3 = %d, n4 = %d, n5 = %d, n6 = %d, n7 = %d, n8 = %d, n9 = %d
   \n", n0, n1, n2, n3, n4, n5, n6, n7, n8, n9);

}
```

3. 设有整型变量 x 和 y 的值分别为 5 和 110。试指出执行完以下循环语句后，变量 x 和 y 的值分别是多少？

（1）while(x<=y)x *=2;

（2）do{x=y/x;y=y-x;}while(y>=1);

(1) x=160; y = 110;

(2) x=18; y=0;

4. 水仙花数是一个 n（n≥3）位数字的数，它等于每个数字的 n 次幂之和。例如，153 是一个水仙花数，$153=1^3+5^3+3^3$。试编写一个程序求小于 999 的所有水仙花数。

```c
#include <stdio.h>

int main()
{
   int x;
   int a, b, c;

   for(x = 100; x <=999; ++x)
   {
      a = x % 10;
      b = (x/10) % 10;
      c = x / 100;

      if(x == a*a*a + b*b*b + c*c*c)
        printf("%d is a narcissus number\n", x);
    }

    return 0;



}
```

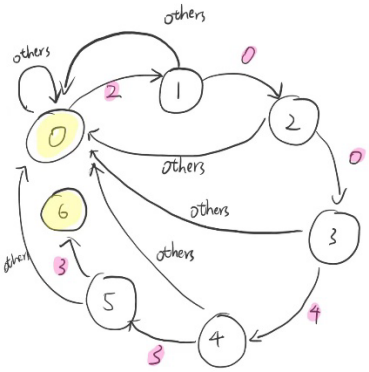
作业 4：用 while, do while 及 for 循环分别完成教材 P65 习题 7

```c
#include <stdio.h>

int main()
{
    int i;

    for(i = 1; i <= 10; i++)
      printf("%d^2=%d, %d^3=%d\n", i, i*i, i, i*i*i);

    i = 1;
    while(i <= 10)
    {
       printf("%d^2=%d, %d^3=%d\n", i, i*i, i, i*i*i);
       ++i;
    }

    i = 1;
    do{
       printf("%d^2=%d, %d^3=%d\n", i, i*i, i, i*i*i);
       ++i;
    } while(i<=10);

   return 0;

}
```

作业 5：利用状态机的方式，识别串"200433"

```c
#include <stdio.h>
int main() {
    int input, state = 0;
    while (true) {
        scanf("%d", &input);
        switch (state) {
            case 0: if (input == 2) state = 1; else state = 0; break;
            case 1: if (input == 0) state = 2; else state = 0; break;
            case 2: if (input == 0) state = 3; else state = 0; break;
            case 3: if (input == 4) state = 4; else state = 0; break;
            case 4: if (input == 3) state = 5; else state = 0; break;
            case 5: if (input == 3) state = 6; else state = 0; break;
            default: state = 0;break;
        }
        if (state == 6) {
            printf("correct\n");
        break;
        }
    }
}
```

以下是本题状态机的状态跳转图，可能大家还没有学到过这种表示方式，但可以自己先悟一悟，圆圈里的是状态机的状态 state，箭头上的数字代表输入，箭头方向代表输入该数字后状态机会跳转到的下一个状态，0 为初始态，6 为终态。习题课的时候会给大家专门讲一讲状态机


others
others
0
2
1
2
0
0
others
others
6
others
3
others
3
5
4
4
5

第五次作业解析

15. 编写程序，按下面的公式计算 e^x 的值。

$$e^x=1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\cdots$$

```c
#include <stdio.h>

int main()
{
  double e, x, t;
  x = 1.0;
  t = e = 1.0;

  int i;

  for(i = 1; i <= 20; ++i)
    {
      t *=  x / i;
      e += t;
    }

  printf("%f\n", e);

  return 0;
}
```

本题思路是通过 t 来表示每个单项式的值，可以注意到第 n 个单项式是第 n-1 个单项式的 x/n 倍，所以不需要从头计算每个单项式分子和分母的值。当然，也可以在计算每个单项式时都用 pow(x,n)计算分子，然后用一个循环计算 n!，但是这样的计算量会大很多。

20. 回文整数是指正读和反读相同的整数，编写一个程序，输入一个整数，判断它是否是回文整数。

```c
#include <stdio.h>

int main()
{
    int d;
    int a;
    int t;
    scanf("%d", &d);

    a = 0;
    t = d;
    while(t > 0)
    {
       a = a*10 + t % 10;
       t /= 10;
    }

    if(a == d)
    {
      printf("Yes\n");
    }
    else
    {
      printf("No\n");
    }

}

```

第 12-16 行的作用是将整型变量 d 的值颠倒后存储到 a 中

22. 草地上有一堆野果，有一只猴子每天去吃掉这堆野果的一半又一个，5 天后刚好吃完这堆野果。求这堆野果原来共有多少个？猴子每天吃多少个野果？

```c
#include <stdio.h>

int main()
{
   int total = 2;
   int t;
   int i;
   printf("Day 5: 2\n");
   for(i = 0; i < 4; i++)
   {
      t = total;
      total = (total + 1) * 2;
      printf("Day %d: %d\n", 4-i, total - t);

   }
   printf("Total: %d\n", total);

   return 0;


}

```

23. 输入自然数 n（n>1），输出该数的质因子分解式。例如，n=30，程序输出 30=2*3*5

```c
#include <stdio.h>

int main()
{
   int d, i;
   scanf("%d", &d);

   printf("%d=", d);
   while(d > 1)
   {
      for(i = 2; i <= d; ++i)
      {
         while(d % i == 0)
         {
             printf("%d", i);
             if(d != i) printf("*");
             d /= i;
         }
      }
   }
   printf("\n");
   return 0;
}
```

不需要判断 i 是否为质数，因为若 i 为合数，则 d 在之前的循环中已经被 i 的质因子整除过了，于是 d%i 一定不为 0 。当 d==i 时说明这是该数分解得到的最后一个质因子，所以不需要在分解式末尾添加乘号了。

24. 已知自变量 x 在区间[0,3]上，函数 $f(x)=x^3-x^2-1$ 有一个实根，试用二分法求该函数的根。

```c
#include <stdio.h>

int main()
{
   double lower, upper, middle, sol;
   lower = 0.;
   upper = 3.;

   while(upper - lower > 1e-5)
   {
       middle = (lower + upper) / 2.0;
       if(middle * middle * middle - middle * middle - 1.0 > 0)
         upper = middle;
       else
         lower = middle;
   }
   sol = (upper + lower) / 2.0;
   printf("Solution = %f\n", sol);
   printf("Residue = %f\n",  sol * sol * sol - sol * sol - 1.0);

   return 0;

}
```

Residue 是残差，表示在该 solution 下 f(x)的值

第六次作业解析

4. 编写将已知数组内容复制到另一个新数组，使复制产生的新数组包含已知数组全部出现过的值，而又不重复。

```c
#include <stdio.h>

int main()
{
  int a[] = {3, 4, 4, 6, 8, 10, 10, 12, 13};
  int n = sizeof(a) / sizeof(int);
  int b[10];        // 用于存储非重复元素
  int m = 0, i, j; // m代表数组b当前有效元素的个数

  for (i = 0; i < n; ++i)
  {
    for (j = 0; j < m; ++j) // 寻找数组b中是否与a[i]相同的元素
    {
      if (b[j] == a[i])
        break;
    }

    if (j == m)
      b[m++] = a[i]; // j等于m说明内循环完整遍历了一次仍未找到相同元素
  }

  for (i = 0; i < m; ++i)
    printf("%d ", b[i]);

  printf("\n");

  return 0;
}
```

6. 输入两个多项式的各项系数和指数，编写程序求出它们的和，并要求与手写习惯相同的格式输出。规定：一个多项式的输入以输入指数为负数结束。多项式的每一项 ax^b 用 ax^b 格式输出。

此题可以用两种思路来解决：

1. 用两个数组 poly_index 和 poly_coeff 分别存储一个多项式的指数和系数，而后比较两个多项式的 poly_index 中的元素值，将同类项对应的 poly_coeff 中的元素值相加，作为合并后的多项式的系数值。
2. 用一个数组来存储一个多项式的指数和系数，该数组的元素存储方式可表示为 a[多项式中的单项式 b 的指数]=多项式中的单项式 b 的系数，然后将两个多项式对应的数组元素依次相加，即完成多项式的合并

需要注意：

1. 输出的多项式结果中，第一个单项式前没有加号，所以要引入额外的逻辑来判断是否为第一个单项式
2. 若单项式的系数为负数，则不需要在输出单项式前添加加号

思路 1:

```
#include <stdio.h>

void main()
{
	int polya_index[100];
	double polya_coeff[100];
	int polyb_index[100];
	double polyb_coeff[100];
	int polyc_index[100];
	double polyc_coeff[100];
	int num_a;
	int num_b;
	int num_c;

	int i,j,k,t;
	int index;
	double coeff;
	int temp;
	double coeff_temp;

	i = 0;
	printf("Please enter polynomial A\n");
	while(1)
	{
		scanf("%d%lf", &index, &coeff);

		if(index < 0)
			break;
		else
		{
			polya_index[i] = index;
			polya_coeff[i] = coeff;
			i++;
		}
	}
	num_a = i;

	i = 0;
	printf("Please enter polynomial B\n");
	while(1)
	{
		scanf("%d%lf", &index, &coeff);
```

```
        if(index < 0)
            break;
        else
        {
            polyb_index[i] = index;
            polyb_coeff[i] = coeff;
            i++;
        }
}
num_b = i;

// sort polynomial A according to the index
for(i = 0; i < num_a; ++i)
        for(j = num_a-1; j>i; --j)
        {
            if(polya_index[j] < polya_index[j-1])
            {
                temp = polya_index[j];
                polya_index[j] = polya_index[j-1];
                polya_index[j-1] = temp;

                coeff_temp = polya_coeff[j];
                polya_coeff[j] = polya_coeff[j-1];
                polya_coeff[j-1] = coeff_temp;

            }
        }

            // sort polynomial B according to the index
for(i = 0; i < num_b; ++i)
        for(j = num_b-1; j>i; --j)
        {
            if(polyb_index[j] < polyb_index[j-1])
            {
                temp = polyb_index[j];
                polyb_index[j] = polyb_index[j-1];
                polyb_index[j-1] = temp;

                coeff_temp = polyb_coeff[j];
                polyb_coeff[j] = polyb_coeff[j-1];
                polyb_coeff[j-1] = coeff_temp;

            }
```

```
}

// add polynomial A & B
for(i = j = k = 0; i < num_a && j < num_b; )
{
	if(polya_index[i] < polyb_index[j])
	{
		polyc_index[k] = polya_index[i];
		polyc_coeff[k] = polya_coeff[i];
		k++;i++;
	}
	else if(polya_index[i] == polyb_index[j])
	{
		polyc_index[k] = polya_index[i];
		polyc_coeff[k] = polya_coeff[i] + polyb_coeff[j];
		i++;j++;k++;
	}
	else if(polya_index[i] > polyb_index[j])
	{
		polyc_index[k] = polyb_index[j];
		polyc_coeff[k] = polyb_coeff[j];
		k++;j++;
	}
}

if(i != num_a)
{
	for(t = i; t < num_a; ++t)
	{
		polyc_index[k] = polya_index[t];
		polyc_coeff[k] = polya_coeff[t];
		k++;
	}
}
else if(j != num_b)
{
	for(t = j; t < num_b; ++t)
	{
		polyc_index[k] = polyb_index[t];
		polyc_coeff[k] = polyb_coeff[t];
		k++;
	}
}
```

```
        num_c = k;

        // printf the result
        for(i = 0; i < num_c; ++i)
        {
            printf("%.1fx^%d", polyc_coeff[i],polyc_index[i]);
            if(i != num_c -1)
                printf("+");
            else
                printf("\n");
        }


}
```

思路 2:

```
#include <stdio.h>

void main()
{
    int polya_coeff[200];
    int polyb_coeff[200];
    int polyc_coeff[200];

    int index, coeff;
    int i;

    printf("Please enter polynomial A\n");
    while (1)
    {
        scanf("%d%lf", &index, &coeff);

        if (index < 0)
            break;
        else
        {
            polya_coeff[index] = coeff;
        }
    }

    printf("Please enter polynomial B\n");
    while (1)
    {
```

```
            scanf("%d%lf", &index, &coeff);

            if (index < 0)
                break;
            else
            {
                polyb_coeff[index] = coeff;
            }
        }

        // add polynomial A&B
        for (i = 0; i < 200; i++)
            polyc_coeff[i] = polya_coeff[i] + polyb_coeff[i];

        // print the result
        int first_flag = 1; // indeicate whether the first monomial has been printed
        for (i = 199; i >= 0; i--)
        {
            if (polyc_coeff[i] == 0)
                continue;
            if (first_flag)
            {
                printf("%.1fx^%d", polyc_coeff[i], i);
                first_flag = 0;
            }
            else if (polyc_coeff[i] > 0)
            {
                printf("+%.1fx^%d", polyc_coeff[i], i); // '+' operator is needed for printing
            }
            else
            {
                printf("%.1fx^%d", polyc_coeff[i], i); //    no need to print '-' because '-' is already
in the coeff
            }
        }
}
```

7. 采用筛选法求质数。算法思想简述如下：

（1）将数组中下标为 0 和 1 的元素设置为 0，下标为 2～N 的元素都设置为 1。

（2）从下标为 2 的元素开始考查，当发现当前位置的数组元素值为 1 时，将下标是当前下标 2 倍、3 倍、……的那些元素全部置 0。

（3）重复步骤（2），直至考查了数组的全部元素，那些值依旧为 1 的元素的下标都是质数。

```c
#include <stdio.h>

int main()
{
   int a[100];
   int i, j;
   a[0] = a[1] = 0;

   for (i = 2; i < 100; i++)
      a[i] = 1;

   for (i = 2; i < 50; i++)
   {

      // 不需要处理a[i]等于0时的情况，因为若i为合数，则i的倍数已经在此前被判断为合数了
      if (a[i] == 1)
      {
         // 遍历i的倍数，i*j一定为合数
         for (j = 2; i * j < 100; j++)
         {
            a[i * j] = 0;
         }
      }
   }

   for (i = 1; i < 100; i++)
      if (a[i] != 0)
         printf("%d ", i);

   printf("\n");

   return 0;
}

```

第七次作业解析

3. 输入 C 程序源程序正文，找出可能存在的圆括号和花括号不匹配的错误。

```
#include <stdio.h>

void main()
{
	char prog[1000];
	int p;

	char stack[100];
	int top = 0;
	int unmatch = 0;

	stack[top] = 0;

	int i = 0;
	char c;
	while ((c = getchar()) != EOF)
	{
		prog[i++] = c;
	}
	prog[i] = '\0';

	/*int ignore = 0;*/
	for (p = 0; prog[p]; ++p)
	{
		/*if(ignore)
			continue;*/
		switch (prog[p])
		{
		case '(':
		case '[':
		case '{':
			top++; // stack[0] is not used for storing character
			stack[top] = prog[p];
			break;
		case ')':
			if (stack[top] != '(')
			{
				unmatch = 1;
			}
```

```
                else
                        top--;
                break;
        case ']':
                if (stack[top] != '[')
                {
                        unmatch = 1;
                }
                else
                        top--;
                break;
        case '}':
                if (stack[top] != '{')
                {
                        unmatch = 1;
                }
                else
                        top--;
                break;
                /*
                case '\'':
                case '\"':
                        ignore = !ignore;
                        break;
                */

        default:
                break;
        }
        if (unmatch)
                break;
    }

    if (unmatch || top != 0)
        printf("not match\n");
    else
        printf("match\n");
}
```

本题使用堆栈来解决问题，是因为花括号的匹配遵循“越外层的括号距离越远”，而栈有“最先进入的最后出来”的特性，两者相符，所以本题天然地可以用栈来解决。有些同学尝试用了其它方法来解决，但目前没有看到栈以外的其它正确方法。

10. 设数组的每个元素只存储 0～9 的数，把该数组的前 n 个整数的排列看做是一个整数的一种表示。现要求编写程序，对数组中的元素做调整，产生一个新的排列，使新的长整数比调整前的长整数大（如果可能），但又是所有更大的表示中最小的。a[]={3,2,6,5,4,1}，则更大又是最小的排列为{3,4,1,2,5,6}。

```
#include <stdio.h>
#define N 6

int main(int argc, char *argv[])
{
    int a[N] = {3, 2, 6, 5, 4, 1};
    int i, j, k, pos, temp;
    int flag = 1;

    printf("before permuation\n");
    for (k = 0; k < N; k++)
        printf("%d", a[k]);
    printf("\n");

    for (i = N - 2; i >= 0; i--)
    {
        for (j = N - 1; j > i; j--)
        {
            if (a[j] > a[i])
            {
                temp = a[j];
                a[j] = a[i];
                a[i] = temp;
                flag = 0;
                break;
            }
        }
        if (flag == 0)
            break;
    }

    if (i == -1)
        printf("No solution\n");
    else
    {
        for (j = i + 1, k = N - 1; j < k; j++, k--)
        {
            temp = a[j];
            a[j] = a[k];
```

```
                a[k] = temp;
            }
            // 也可以用冒泡排序
            // for (j = i + 1; j < N; ++j)
            //     for (k = N - 1; k > j; k--)
            //     {
            //         if (a[k] < a[k - 1])
            //         {
            //             temp = a[k];
            //             a[k] = a[k - 1];
            //             a[k - 1] = temp;
            //         }
            //     }

            printf("after permuation\n");
            for (k = 0; k < N; k++)
                printf("%d", a[k]);
            printf("\n");
        }
}
```

本题的思路是“在尽量低的位数进行调整”，所以要先从较低的位开始寻找能让数字稍微大一点的调整方案。在找到可交换位之后进行交换，并将其余低位按从小到大顺序排列，即得到刚好大一点的数。

16. 整理字符串，将字符串的前导空白符和后随空白符删除，并将字符串中非空白字符串之间的连续的多个空白符只保留一个，而去掉多余的空白符。

```
#include <stdio.h>

int main()
{
  char str[] = "  I    love fdu    ";
  int p, q;

  p = q = 0;
  // remove the initial spaces
  for (; str[p] == ' '; ++p)
    ;

  for (; str[p]; ++p)
  {
    if (str[p] != ' ')
      str[q++] = str[p];
```

```
    else if (str[p] == ' ')
    {
      if (str[p - 1] != ' ')
        str[q++] = str[p];
    }
  }

  if (str[q - 1] == ' ')
    str[q - 1] = '\0';
  else
    str[q] = '\0';

  printf("%s\n", str);

  return 0;
}
```

注意末尾很容易多出一个空格，而 printf 无法直接看出，可以在占位符%s 之后再加一个字符来检查末尾字符有没有被删除。

编写两个字符串连接的函数 strcat

```
#include <stdio.h>
#include <string.h>
#define N 100

/* or define a sub function */
void strcat(char str1[], char str2[])
{
    int i = 0;
    for (; str1[i]; i++)
        ;
    /* after loop, i point to the '\0' of str1*/
    for (int j = 0; str2[j]; j++, i++)
        str1[i] = str2[j];
    str1[i] = '\0';
    return;
}

int main()
{
    char str1[N], str2[N];
    printf("Input 1st string: ");
    gets(str1);
    printf("Input 2nd string: ");
```

```
    gets(str2);

    int len1 = strlen(str1);
    int len2 = strlen(str2);
    for (int i = 0; i <= len2; i++)
    {
        str1[len1 + i] = str2[i];
    }

    printf("After str cat: ");
    puts(str1);
}
```

这段代码在主函数里实现了 strcat，也单独定义了一个 strcat 函数来实现该功能

第八次作业解析

4. 读程序，指出以下程序的功能。

程序 1：

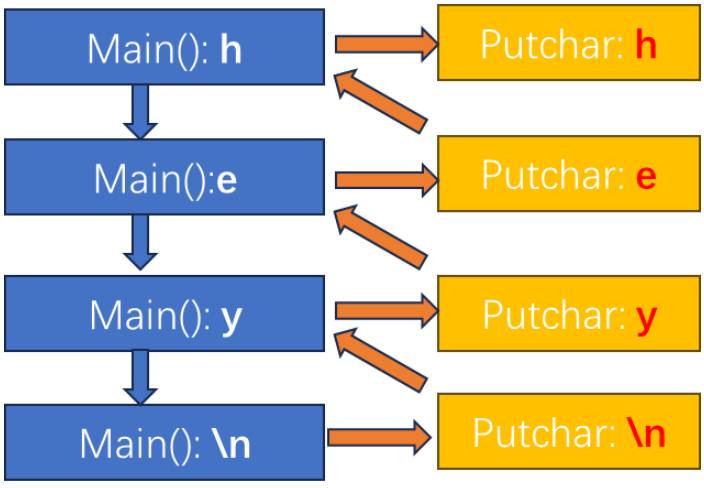
```
#include<stdio.h>
void main()
{  int c;
   if((c=getchar())!='\n')
   {  main();    printf("%c",c);
   } else printf("\n");
}
```

程序 2：

```
#include<stdio.h>
int mul(int,int);
void main()
{  int u,v,w;
   printf("Enter two integers!\n");  scanf("%d%d",&u,&v);
   if(v>=0)  w=mul(u,v);
   else w=-mul(u,-v);
   printf("The result is %d\n",w);


}
int mul(int a,int b)
{  if(a==0||b==0) return 0;
   if(b==1) return a;
   return a+mul(a,b-1);
}
```

程序 1：读入一行字符，然后倒序输出

程序 2：计算两个整数 a b 的乘积



程序 1 ：

main 函数在这里相当于递归函数，由于 main(); 语句的调用再 printf 语句之前，所以先被输入的字符反而在后面才输出，宏观上便是字符串被倒叙输出了。递归的这一特点和栈的“先入后出”特性很像

另外可以尝试用单步调试的方法运行完整个程序，或许能更直观地理解整个过程

程序 2：

利用递归方法“分而治之”的思想，把 a*b 变为“a+a*(b-1)”，于是乘法变为加法。b 为负数时则先计算 a*|b|，然后再对计算结果取相反数。

5. 编写一个函数，用时间时、分和秒作为函数的 3 个形参，函数返回自 0 点钟到指定时间的秒数，并用这个函数计算同一天两个时间之间的秒数。

```c
#include <stdio.h>


int gettime(int hour, int minute, int second)
{
    return hour * 3600 + minute * 60 + second;
}


int main()
{
    int t1 = gettime(8, 30, 1);
    int t2 = gettime(10, 40, 2);

    printf("10:40:2 - 8:30:1 = %d\n", t2 - t1);
    return 0;
}

```

6. 如果一个整数（整数>1）的各因子（包括 1，但不包括整数自身）之和等于该整数，称这样的整数为完全数。例如，因为 6=1+2+3，所以 6 是完全数。编写一个已知整数判断其是否是完全数的函数，并用该函数输出 1 000 之内的所有完全数。

```c
#include <stdio.h>


int complete_number(int n)
{
   int y = 0;

   int m = 0;
   int i;
   for(i = 1; i < n-1; i++)
      if(n % i == 0) m += i;

   return m == n;
}

int main()
{
   int i;

   for(i = 1; i < 1000; i++)
     if(complete_number(i))
        printf("%d\n", i);

   return i;

}

```

return m==n 意思是返回"m==n"的真假。如果成立则为真，即 return 1，反之则为假，return 0

9. 试用递归函数，返回与所给十进制整数相反顺序的整数，如已知整数是 1 234，函数返回值是 4 321。

```c
#include <stdio.h>
#include <math.h>

int reverse(int n, int m)
{
    if(m == 1) return n;

    return (n % 10) * pow(10, m-1) + reverse(n/10, m-1);
}

int main()
{
    int n;
    int m = 0;

    scanf("%d", &n);

    int i = n;
    while(i)
    {
        i /= 10;
        m++;
    }

    int t = reverse(n, m);

    printf("%d\n", t);

    return 0;
}
```

T9:

构造如下等式：一个数的颠倒数 = 把它的最低位乘到最高位 + 剩余位数的颠倒数

“大”问题 = 子问题 + “小”问题

分而治之

一个数的颠倒数=把它的最低位乘到最高位+剩余位数的颠倒数，这个等式两边都有“颠倒数”这一个需要我们编写的函数，而等式左边的颠倒数比等式右边的颠倒数是一个“更大”，因此一个大问题被拆分为了两个小问题，这符合递归“分而治之”的思想，我们就可以根据这个思路来编写递归函数了。事实上，所有的递归题目都可以采用构造这种等式的方法来得到思路。

递归函数的参数可以有多个，本题的 reverse 函数中，n 代表需要逆序的整数，而 m 代表 n 的位数。

仅用一个参数 n 也可以实现操作，但每次都需要判断 n 的位数，引入了许多冗余操作，这与递归“简洁优雅直观”的天性不符

第九次作业解析

2. 举例说明指针的定义和引用指针及引用指针所指变量的方法？
3. 举例说明指针与数组名之间的联系与区别。
6. 什么情况下需要为函数设置指针形参？

T2．

①指针定义

```
int *p;
```

②引用指针

```
int a;
int *p;
p = & a;
```

③引用指针所指向变量

```
*p = 5;
```

T3.

指针和数组名的联系：

两者都是地址，指针存储地址，数组名代表数组首地址。作为函数形参时，两者等同，都可以通过方括号[]来访问相应数据。

指针和数组名的区别：

指针是变量，数组名是常量。

T6

① 需要借助指针形参来间接改变调用环境中的变量的值（如果要修改调用环境中的一级指针指向的地址，则形参为一级指针的指针，即为二级指针，以此类推）
② 传输的数据是数组时

10. 有若干个学生，每人考 4 门课程，设用二维数组存储学生的成绩，二维数组的一行对应一个学生的成绩，每行的第一个数是学生的学号。试以此数据结构为基础，编写两个函数：一个是已知成绩表和学号，返回该生成绩表的函数；另一个是已知某个学生表，输出学生学号和成绩的函数。要求两个函数采用指针编写。

```c
#include <stdio.h>


int * search_by_id(int scores[][5], int n, int id)
{
   int i;
   for(i = 0; i < n; ++i)
     if(id == scores[i][0])
       return scores[i];

   return NULL;
}

void print_scores(int scores[][5], int n)
{
   int i;
   for(i = 0; i < n; i++)
   {
      printf("%d: %d, %d, %d, %d\n", scores[i][0],
             scores[i][1], scores[i][2], scores[i][3], scores[i][4]);
   }
}


int main()
{
  int a[100][5];

  int i;
  for(i = 0; i < 100; i++)
  {
     a[i][0] = i;
     a[i][1] = a[i][2] = a[i][3] = a[i][4] = 90;
  }

  int *p;

  p = search_by_id(a, 100, 50);
  printf("%d: %d, %d, %d, %d\n", p[0], p[1], p[2], p[3], p[4]);
  print_scores(a, 100);

  return 0;
}
```

两个函数的参数中的 int scores[][5]也可定义为:

1. 指向二维数组的指针，即 int (*scores)[5]，引用第 i 行第 j 列元素的方式也为 scores[i][j]
2. 一维指针，即 int *scores，此时参数中还要包含行数 int n，列数 int m，引用第 i 行第 j 列元素的方式则为 *(scores + i * m + j)

11. 用指针编写在数组中查找指定值指针的函数。

```c
#include <stdio.h>

int *search(int *a, int n, int id)
{
    int i;
    for (i = 0; i < n; i++)
        if (id == a[i])
            return &a[i];

    return NULL;
}

int main()
{
    int a[100];
    int *p;
    int i;
    for (i = 0; i < 100; i++)
        a[i] = i;

    p = search(a, 100, 40);

    printf("target value is: %d\n", *p);
    printf("address of the value is: %lx", p);

    return 0;
}

```

返回值也可写为 return a+i;
Printf 目标值所在地址时，占位符要使用 long int 类型占位符，因为目前主流 PC 的地址长度都为 64bit，对应一个 long int，使用%d 占位符可能会向上溢出。另外，一般情况下我们使用八进制/十六进制来表示地址，所以占位符一般会使用%lx 或%lo（用%d 也没问题）。

在&、*、[]比较多的情况下，判断变量类型时可能会被绕晕，可以采取这样的方法：想象有一个*计数器，这个变量每与一个&组合就加一个*，每与一个[]或者*组合就减一个*，最后再统计有多少个*，就代表这个变量需要用几级指针来存储。
例如本题 search 函数中的 int *a，*计数器原本有一个*，写成&a[i]之后，遇到&加了一个*，遇到[i]又减了一个*，所以最后*计数器还是剩一个*，于是&a[i]依然是一个 int *类型的值。
对于一维数组的数组名，*计数器初始有一个*，二维数组的数组名则有两个*，二维数组的指针也有两个*（因为二维数组指针本质上是一个二级指针），以此类推。
在做指针类型题目时很容易遇到变量类型不匹配的问题，可以通过*计数器的方式来进行检查，找到问题的效率就会比较高。

拓展

- 星号*计数器 cnt
    - 变量的定义决定了它初始有几个*
        - 变量类型里每多一个*或者方括号[]，则 cnt 加 1
            - 0 个: char a, int a
            - 1 个: int *b, int b[10]
            - 2 个: int **c, int c[10][10], int *c[10], int (*c)[10]
    - 变量与&、*、[]结合为表达式，cnt 发生改变
        - 表达式每多一个*或者[]，cnt 减 1，每多一个&，cnt 加 1
        - 例如，定义变量 int a, int *b, int (*c)[10];，将 a b c 与&*[]结合后，对应 cnt 数量如下
            - 0 个: a, *b, *(c+1)[5]
            - 1 个: b, &a, c[1], (*c+1), &b[0]
            - 2 个: c, &b, &c[1]
    - 最后根据 cnt 的值确定表达式对应的变量类型
        - 0 个: 对应具体的值
        - 1 个: 对应某个地址
        - 2 个: 对应某一行的地址（当变量为二维数组指针时）

第十次作业解析

14. 写出以下函数的功能。

```
char *strpos(char *s,char *t)
{ char *j,*k;
  for(;*s;s++)
  { for(j=s,k=t;*k&&*j==*k;j++,k++);
    if(k!=t&&*k=='\0')  return s;
  }
  return NULL;}
```

功能：查找字符串 t 在 s 中的位置。

注意内层的 for 循环后面有个分号，所以 if 语句实际上并不是内循环的语句块。对于内循环，如果 s 中的这个子字符串刚好与字符串 t 相同，这个 for 循环就会一直进行下去，直到 k 指向了 t 的结束符\0，所以判断 for 循环之后 k 有没有指向结束符\0，就能够判断字符串 t 是否与 s 的当前子字符串相等了。

17. 用指针描述以下计算：

（1）将方阵的上三角元素与下三角对应元素交换，使方阵沿着主对角线转 180° 。

（2）字符串中的字符个数。

（3）将字符串 s2 中的前 n 个字符复制到字符数组 s1[]。

（4）将字符串 s2 中的字符复制连接在字符数组 s1[]中字符串之后，使它存有更长的字符串。

（5）将字符串 s 中的小写英文字母改写成大写英文字母。

（6）整理字符串，将字符串中前导和后随的空白符删除，字符串中间连续的多个空白符只保留一个，去掉多余的空白符。

（1）方阵元素交换

```c
#include <stdio.h>

void swap(int a[][10])
{
   int i, j, temp;
   for(i = 0; i < 10; i++)
     for(j = 0; j < i; j++)
        {
           temp = a[i][j];
           a[i][j] = a[j][i];
           a[j][i] = temp;
        }
}

int main()
{
    int a[10][10];

    int i, j, k = 0;
    for(i = 0; i < 10; i++)
    {
       for(j = 0; j < 10; j++)
         {
           a[i][j] = k++;
           printf("%d ", a[i][j]);
         }
       printf("\n");
    }
    swap(a);

    for(i = 0; i < 10; i++)
    {
      for(j = 0; j < 10; j++)
        printf("%d ", a[i][j]);

      printf("\n");
    }
    return 0;
}
```

Swap 函数内循环的循环条件是 j<i 而不是 j<10，如果写成 j<10，那每个元素都会被交换两次，等于没有交换。

（2）字符串的字符个数

```c
#include <stdio.h>

int strlen(char *s)
{
   int i = 0;
   while(*s)
    {
        ++i; ++s;
    }
   return i;
}


int main()
{
   char s[] = "China";
   printf("length of %s is %d\n", s, strlen(s));
   return 0;
}
```

（3）将 s2 的前 n 个字符复制到 s1

```c
#include <stdio.h>

void strncpy(char s1[], char s2[], int n)
{
    int i;
    for(i = 0; i < n; i++)
       s1[i] = s2[i];

}

int main()
{
    char s1[10];
    char s2[] = "China_1";

    strncpy(s1, s2, 5);

    printf("%s\n", s1);

    return 0;
}
```

（4）将 s2 的字符复制连接到 s1 的字符串后

```c
#include <stdio.h>


void strcat(char s1[], char s2[])
{
   char *s = s1;
   char *t = s2;
   while(*s)
      s++;

   while(*t)
   {
      *s++ = *t++;
   }
   *s = 0;

}

int main()
{
   char s1[20] = "China ";
   char s2[10] = "Beijing";

   strcat(s1, s2);

   printf("%s\n", s1);

   return 0;

}
```

（5）小写字母改为大写字母

```c
#include <stdio.h>

void strupper(char s[])
{
    char *p = s;
    while(*p)
    {
       if(*p >= 'a' && *p <= 'z')
          *p -= ('a' - 'A');
       p++;
    }
}

int main()
{
   char s[] = "China";
   strupper(s);
   printf("%s\n", s);

   return 0;
}
```

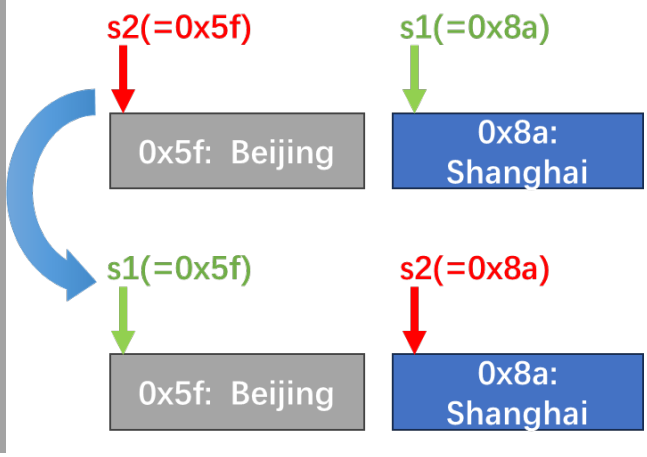
（6）删除字符串前导和后随空格，单词间只保留一个空格

```c
#include <stdio.h>

int main()
{
   char p[256];
   gets(p);

   char * s = p;
   char * t = p;

   for(; *s == ' '; s++);

   for(; *s; s++)
   {
      if(*s != ' ' || (*s == ' ' && *(s-1) != ' '))
         *(t++) = *s;
   }

   if(*(t-1) == ' ') *(t-1) = 0;
   else *t = 0;

   puts(p);

   return 0;
}
```

19. 编写用引用形参实现两个字符串交换的函数。例如，调用函数前有：

```c
char *s1="Shanghai";    char *s2="Beijing";
```

```c
#include <stdio.h>

void swap(char** s1, char **s2)
{
   char *t;
   t = *s1;
   *s1 = *s2;
   *s2 = t;
}

int main()
{
   char *s1 = "Shanghai";
   char *s2 = "Beijing";

   swap(&s1, &s2);

   printf("%s, %s\n", s1, s2);

   return 0;

}
```



该题的本意是交换指针指向的字符串地址，而不是要改变字符串本身的内容（按照题目定义字符串的方式，"Shanghai"和"Beijing"是字符串常量，无法通过指针修改值）。改变 s1 和 s2 的值（即指向的地址），就需要把 s1 和 s2 的地址作为参数传递进 swap 函数，因此参数的类型是二级指针而不是一级指针。

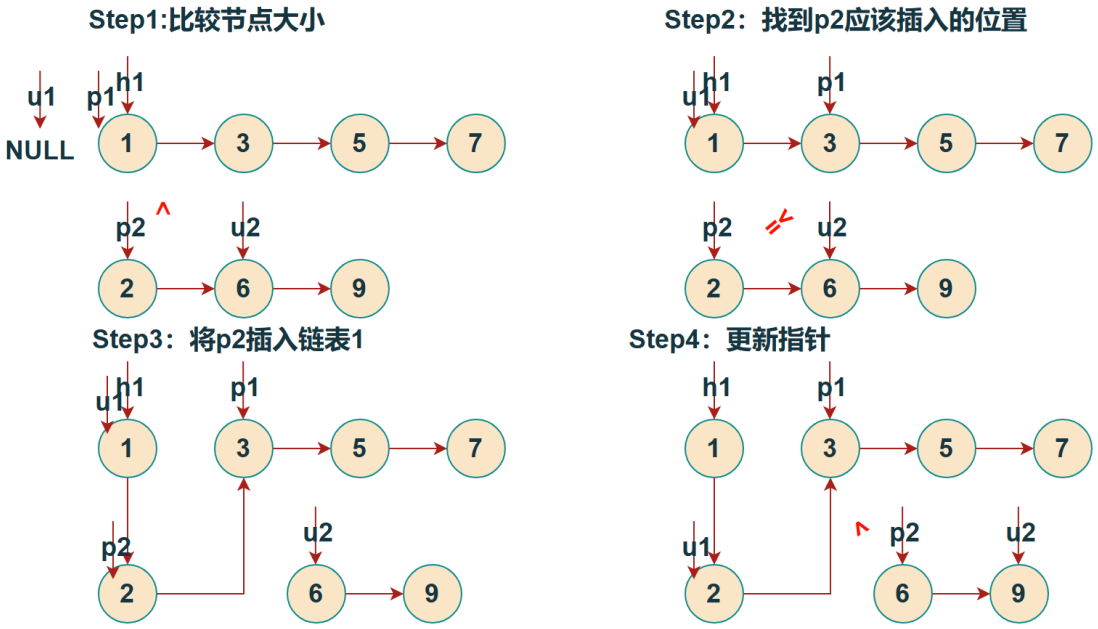
第十一次作业解析

1. 自己设计一个通信录条目的结构，并分别编写输入和输出一条通信录条目的函数。

```c
#include <stdio.h>

struct rec
{
   char name[20];
   int tel;
};

void input(struct rec *r, int n)
{
   int i;
   for (i = 0; i < n; i++)
   {
      scanf("%s", r[i].name);
      scanf("%d", &r[i].tel); // 注意取地址符
   }
}

void output(struct rec *r, int n)
{
   int i;
   for (i = 0; i < n; i++)
      printf("%s, %d\n", r[i].name, r[i].tel);
};

int main()
{
   struct rec record[2]; // 定义结构体数组

   input(record, 2); // 数组名代表结构体数组首个元素的地址，2代表数组大小
   output(record, 2);

   return 0;
}
```

注意在 visual studio 中，用 scanf_s 函数输入字符串时，需要添加一个整型参数用于表示缓冲区大小，例如 scanf_s("%s",r[i].name,20); 一般情况下，缓冲区大小可设置为数组的长度

7. 编写实现将两个已知的有序链表合并成一个有序链表的函数。



```c
#include <stdio.h>
#include <stdlib.h>

struct intNode
{
    int data;
    struct intNode *next;
};

struct intNode *headInsert(int n)
{
    int i;
    struct intNode *h = NULL;
    struct intNode *p;

    for (i = 0; i < n; ++i)
    {
        p = (struct intNode *)malloc(sizeof(intNode));
        scanf("%d", &(p->data));

        p->next = h;
        h = p;
    }

    return h;
}

struct intNode *tailInsert(int n)
{
    int i;
    struct intNode *h = NULL;
    struct intNode *tail;
    struct intNode *p;

    for (i = 0; i < n; ++i)
    {
        p = (struct intNode *)malloc(sizeof(struct intNode));
        scanf("%d", &(p->data));

        if (h == NULL)
            h = tail = p;
        else
        {
            tail->next = p;
```

```
            tail = p;
        }
    }
    tail->next = NULL;

    return h;
}

void print(struct intNode *h)
{
    struct intNode *p = h;
    for (; p; p = p->next)
    {
        printf("%d ", p->data);
    }
    printf("\n");
}

// 逐个节点释放空间
void free_data(struct intNode *h)
{
    struct intNode *p = h;
    struct intNode *u;

    for (; p;)
    {
        u = p->next;
        free(p);
        p = u;
    }
}

// 排序的思路：遍历原链表 h 的每一个节点，把该节点取出，并插入新链表合适的位置
struct intNode *sort(struct intNode *h)
{
    struct intNode *p = h, *u;
    struct intNode *h2 = NULL;
    struct intNode *p2, *u2;

    while (p)
    {
        u = p->next;
        p2 = h2;
        while (p2 && p2->data < p->data)
```

```
        {
            u2 = p2;
            p2 = p2->next;
        }
        if (p2 == h2)
        {
            p->next = h2;
            h2 = p;
        }
        else
        {
            u2->next = p;
            p->next = p2;
        }
        p = u;
    }
    return h2;
}

struct intNode *remove(struct intNode *h, int val)
{
    struct intNode *p, *u, *w;
    p = h;

    while (p)
    {
        if (p->data == val)
        {
            if (p == h)
            {
                h = p->next;
                free(p);
                p = h;
            }
            else
            {
                u->next = p->next;
                free(p);
                p = u->next;
            }
        }
        else if (p->data < val)
        {
            u = p;
```

```
            p = p->next;
        }
        else
            break;
    }

    return h;
}

// 合并的思路：将 h2 的节点逐个拿出，插入 h1 链表合适的位置，最后 h1 指向的即为合并的链表
struct intNode *merge(struct intNode *h1, struct intNode *h2)
{
    struct intNode *p1, *p2, *u1, *u2;
    p1 = h1;
    p2 = h2;

    while (p2 && p1)
    {
        u2 = p2->next;
        if (p2->data <= p1->data)
        {
            if (p1 == h1)
            {
                p2->next = h1;
                h1 = p2;
                u1 = p2;
            }
            else
            {
                u1->next = p2;
                p2->next = p1;
                u1 = p2;
            }
            p2 = u2;
        }
        else
        {
            u1 = p1;
            p1 = p1->next;
        }
    }

    // 若 h2 链表仍剩余节点，则直接嫁接到 h1 链表的末尾
```

```
    if (p2)
    {
        u1->next = p2;
    }

    return h1;
}

int main()
{
    struct intNode *h1, *h2;
    // 利用头部插入法创建有 4 个表元的链表
    h1 = headInsert(4);
    // 对 h1 指向的链表进行排序
    // 本题默认链表已经是有序的，排序并不必要，可以学习 sort 函数的写法
    h1 = sort(h1);
    print(h1);

    h2 = headInsert(4);
    h2 = sort(h2);
    print(h2);

    // 合并两条有序链表
    h1 = merge(h1, h2);
    print(h1);

    free_data(h1);

    return 0;
}
```

几个需要注意的地方：

1. 不要随意移动指向首个表元的指针 h，因为链表是单向的，一旦丢失了首个表元的地址，这个链表就丢失了一部分节点。如果需要遍历操作链表，则首先定义指针 p=h，然后再进行操作
2. 进行插入、删除、排序等操作时，需要判断当前操作的表元是否是头表元，如果头表元的地址发生了改变，需要及时改变 h 指针。并且，由于 h 指向的地址可能变化，因此函数最后要 return h;把头表元地址返回给主函数，然后主函数及时更新链表的头表元地址
3. 使用 malloc 申请节点空间时，使用 malloc(sizeof(struct mystruct))，避免写成 malloc(sizeof(struct mystruct *))，后者申请的空间只有一个指针变量的大小，可能产生内存错误
4. free(p)只会释放 p 节点指向的表元，而不会释放掉整个链表的内存。释放整个链表的内存，需要循环遍历每个节点以进行 free

第十二次作业解析

4. 编写输入学生某门课程成绩的函数，利用该函数输入学生的全部成绩。再编写按某门成绩或总成绩排序的函数。

```c
#include <stdio.h>
#include <stdlib.h>
#define SUBjECTS 3

struct scores_rec
{
    int id;
    char name[20];
    int scores[SUBjECTS];
    int total;
    struct scores_rec *next;
};

struct scores_rec *input()
{
    struct scores_rec *p = (struct scores_rec *)malloc(sizeof(struct
scores_rec));
    scanf("%d", &p->id);
    scanf("%s", p->name);
    int total = 0;
    int i;
    for (i = 0; i < SUBjECTS; i++)
    {
        scanf("%d", &p->scores[i]);
        total += p->scores[i];
    }
    p->total = total;
    return p;
}

struct scores_rec *sort(struct scores_rec *h, int subject)
{
    struct scores_rec *h1 = NULL;       // h1 表示排序后的链表的头表元地址
    struct scores_rec *u1, *w1, *p, *u; // w1 表示 u1 前的表元

    p = h;

    while (p)
```

```
    {
        // 指针 u 记录 p 指向的下一个表元地址，以防 p 变动过程中链表丢失
        u = p->next;
        u1 = h1;
        // 判断按照哪门科目排名
        if (subject == 0)
            // 有序链表节点大于有当前处理的节点则继续向后遍历
            while (u1 && u1->total > p->total)
            {
                w1 = u1;
                u1 = u1->next;
            }
        else
            while (u1 && u1->scores[subject - 1] > p->scores[subject -
1])
            {
                w1 = u1;
                u1 = u1->next;
            }

        // 头表元的处理
        if (u1 == h1)
        {
            p->next = u1;
            h1 = p;
        }
        // 将 P 指向的表元插入有序链表
        else
        {
            w1->next = p;
            p->next = u1;
        }
        // 更新 p 指向下一个节点
        p = u;
    }

    return h1;
}

int main()
{
    struct scores_rec *h, *p;
    h = NULL;
    int i, n, sub;
```

```
    printf("Input student number:");
    scanf("%d", &n);
    printf("Input student information:\n");
    for (i = 0; i < n; i++)
    {
        p = input();
        p->next = h;
        h = p;
    }
    // 输入排序参照的科目
    printf("Input sorting based subject(0 for total):");
    // 第二个参数代表按照哪门课排名，0 代表总分，其它代表单科
    scanf("%d", &sub);
    h = sort(h, sub);

    p = h;
    // 输出排序后的成绩表
    printf("After sorting:\n");
    printf("ID\tNAME\tTOTAL");
    for (int j = 0; j < SUBjECTS; j++)
        printf("\tSUBJ%d", j + 1);
    printf("\n");
    while (p)
    {
        printf("%d\t%s\t%d", p->id, p->name, p->total);
        for (int j = 0; j < SUBjECTS; j++)
            printf("\t%d", p->scores[j]);
        printf("\n");
        p = p->next;
    }
    return 0;
}
```

将无序链表排序为有序链表，可以参考子豪学长录制的链表讲解视频

8. 编写 3 个链表复制函数。第 1 个是复制出相同链接顺序的链表；第 2 个是复制出链接顺序相反的链表；第 3 个是复制出有序链表。

```
#include <stdio.h>
#include <stdlib.h>

struct intNode
{
   int data;
   struct intNode *next;
};

struct intNode *headInsert(int n)
{
   int i;
   struct intNode *h = NULL;
   struct intNode *p;

   for (i = 0; i < n; ++i)
   {
      p = (struct intNode *)malloc(sizeof(intNode));
      scanf("%d", &(p->data));

      p->next = h;
      h = p;
   }

   return h;
}

struct intNode *tailInsert(int n)
{
   int i;
   struct intNode *h = NULL;
   struct intNode *tail;
   struct intNode *p;

   for (i = 0; i < n; ++i)
   {
      p = (struct intNode *)malloc(sizeof(struct intNode));
      scanf("%d", &(p->data));

      if (h == NULL)
         h = tail = p;
      else
```

```
      {
         tail->next = p;
         tail = p;
      }
   }
   tail->next = NULL;

   return h;
}

void print(struct intNode *h)
{
   struct intNode *p = h;
   for (; p; p = p->next)
   {
      printf("%d ", p->data);
   }
   printf("\n");
}

void free_data(struct intNode *h)
{
   struct intNode *p = h;
   struct intNode *u;

   for (; p;)
   {
      u = p->next;
      free(p);
      p = u;
   }
}

struct intNode *sort(struct intNode *h)
{
   struct intNode *p = h, *u;
   struct intNode *h2 = NULL;
   struct intNode *p2, *u2;

   while (p)
   {
      u = p->next;
      p2 = h2;
      while (p2 && p2->data < p->data)
```

```
      {
         u2 = p2;
         p2 = p2->next;
      }
      if (p2 == h2)
      {
         p->next = h2;
         h2 = p;
      }
      else
      {
         u2->next = p;
         p->next = p2;
      }
      p = u;
   }
   return h2;
}

struct intNode *remove(struct intNode *h, int val)
{
   struct intNode *p, *u, *w;
   p = h;

   while (p)
   {
      if (p->data == val)
      {
         if (p == h)
         {
            h = p->next;
            free(p);
            p = h;
         }
         else
         {
            u->next = p->next;
            free(p);
            p = u->next;
         }
      }
      else if (p->data < val)
      {
         u = p;
```

```
         p = p->next;
      }
      else
         break;
   }

   return h;
}

struct intNode *merge(struct intNode *h1, struct intNode *h2)
{
   struct intNode *p1, *p2, *u1, *u2;
   p1 = h1;
   p2 = h2;

   while (p2 && p1)
   {
      u2 = p2->next;
      if (p2->data <= p1->data)
      {
         if (p1 == h1)
         {
            p2->next = h1;
            h1 = p2;
         }
         else
         {
            u1->next = p2;
            p2->next = p1;
            u1 = p2;
         }
         p2 = u2;
      }
      else
      {
         u1 = p1;
         p1 = p1->next;
      }
   }

   while (p2)
   {
      u2 = p2->next;
      u1->next = p2;
```

```
      u1 = p2;
      p2 = u2;
   }

   return h1;
}

// 遍历原链表的同时使用尾部插入法构建新链表即可
struct intNode *copy_same(struct intNode *h)
{
   struct intNode *h1, *t1, *p1, *p;

   h1 = t1 = NULL;
   p = h;
   while (p)
   {
      p1 = (struct intNode *)malloc(sizeof(struct intNode));
      p1->data = p->data;
      if (h1 == NULL)
         h1 = t1 = p1;
      else
      {
         t1->next = p1;
         t1 = p1;
      }

      p = p->next;
   }
   t1->next = NULL;

   return h1;
}

// 遍历原链表的同时使用头部插入法构建新链表即可
struct intNode *copy_reverse(struct intNode *h)
{
   struct intNode *h1, *p1, *p;

   h1 = NULL;
   p = h;
   while (p)
   {
      p1 = (struct intNode *)malloc(sizeof(struct intNode));
      p1->data = p->data;
```

```
      p1->next = h1;
      h1 = p1;

      p = p->next;
   }
   return h1;
}

// 注意与第 4 题的区别：T4 是对原节点排序，不会 malloc 新节点，而这里需要 malloc 以进行“复制”排序
struct intNode *copy_sort(struct intNode *h)
{
   struct intNode *h1, *p1, *u1, *p, *w1;

   h1 = NULL;
   p = h;
   while (p)
   {
      p1 = (struct intNode *)malloc(sizeof(struct intNode));
      p1->data = p->data;

      u1 = h1;
      while (u1 && u1->data < p1->data)
      {
         w1 = u1;
         u1 = u1->next;
      }

      if (u1 == h1)
      {
         p1->next = h1;
         h1 = p1;
      }
      else
      {
         p1->next = u1;
         w1->next = p1;
      }

      p = p->next;
   }
   return h1;
}
```

```c
int main()
{
   struct intNode *h, *h1, *h2, *h3;
   h = headInsert(4);
   print(h);

   h1 = copy_same(h);
   h2 = copy_reverse(h);
   h3 = copy_sort(h);
   print(h1);
   print(h2);
   print(h3);

   free_data(h);
   free_data(h1);
   free_data(h2);
   free_data(h3);

   return 0;
}
```

注意“复制”链表的意思是不改变原链表，因此新链表的表元一定是通过 malloc 申请的。而 T4 这样的题目只是对原链表的表元顺序进行调整，不会创建新的表元，因此操作时不会用到 malloc。

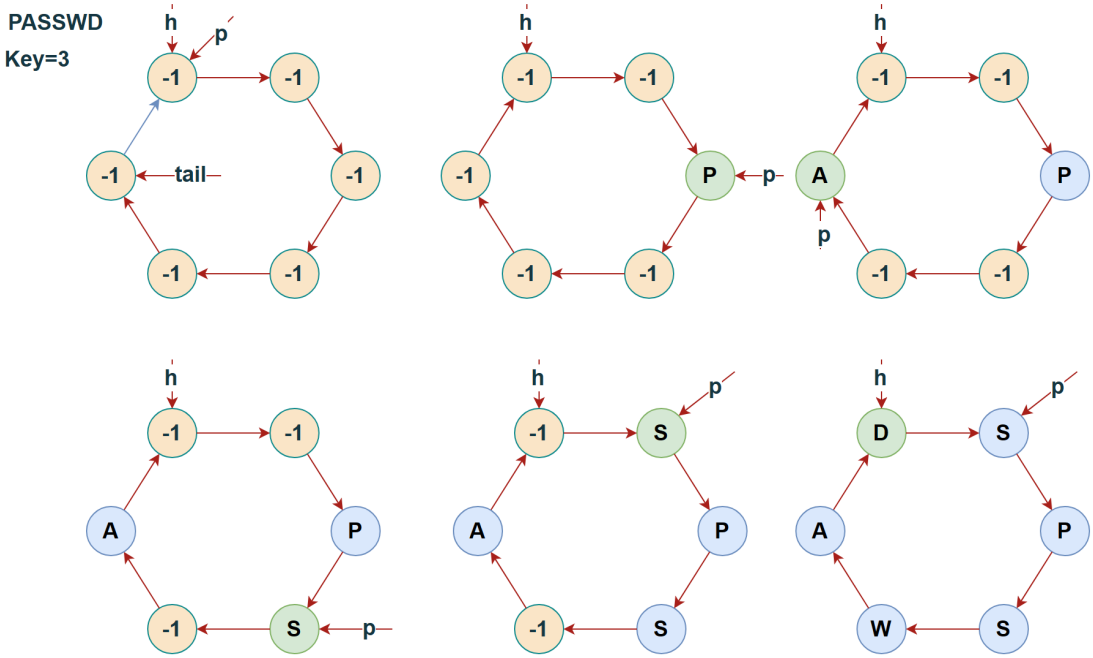
第十三次作业解析

11. 编写从无序的整数链表中找出最小表元，并将它从链表中删除的函数。

```
struct intNode *remove_smallest(struct intNode *h)
{
   struct intNode *u, *w, *us, *ws;
   u = h;
   if (h == NULL)
      return h;
   int smallest = h->data; // 用头表元的值初始化 smallest

   while (u)
   {
      if (u->data < smallest)
      {
         us = u;
         ws = w;
         smallest = u->data;
         // us 记录当前最小表元地址，ws 记录 us 前一个表元地址
      }
      w = u;
      u = u->next;
   }

   // 删除最小表元
   if (us == h)
   {
      h = us->next;
      free(us);
   }
   else
   {
      ws->next = us->next;
      free(us);
   }

   return h;
}
```

13. 试编写按以下加密规则对指定的加密钥匙 key 和原文字符串的加密函数。设原文字符串有 n 个字符，生成的密文字符串也是 n 个字符。将密文字符串的 n 个字符位置按顺时针连成一个环。加密时，从环的起始位置起顺时针方向计数，每当数到第 key 个字符位置时，从原文字符串的第 1 个字符开始，依次将原文中的当前字符放入该密文字符位置中，已填入字符的密文字符位置以后不再在环上计数。重复上述过程，直至原文的 n 个字符全部放入密文环中。由此产生的密文环上的字符序列即为原文的密文。

例子：




```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct Node
{
    int flag;
    char c;
    struct Node *next;
};

struct Node *encrypt(char *str, int key)
{
    int n = strlen(str);
    int i, cnt; // cnt 表示当已经前插入的字符数量
    struct Node *h = NULL, *p, *t = NULL;
    for (i = 0; i < n; ++i)
    { // 初始化链表
        p = (struct Node *)malloc(sizeof(struct Node));
        p->flag = 0;
        p->next = h;
```

```
        h = p;
        if (t == NULL)
            t = p;
    }
    t->next = h; // 连接首尾，构成循环链表
    i = 0;
    cnt = 0;
    p = h;
    while (cnt < n) // 循环条件为已插入字符的数量小于 n
    {
        if (p->flag == 0)
            i++; // 未插入字符，计数器加 1
        if (i == key)
        {
            p->c = str[cnt++];
            p->flag = 1;
            i = 0;
        } // 当遍历到第 key 个空白表元时插入字符
        p = p->next;
    }
    t->next = NULL; // 断开循环链表，以便输出
    return h;
}

int main()
{
    char str[] = "FDU loves me(?)";
    struct Node *h, *p;
    h = encrypt(str, 5);

    p = h;
    while (p)
    {
        putchar(p->c);
        p = p->next;
    }
    putchar('\n');

    return 0;
}
```

本题应注意以下几点：

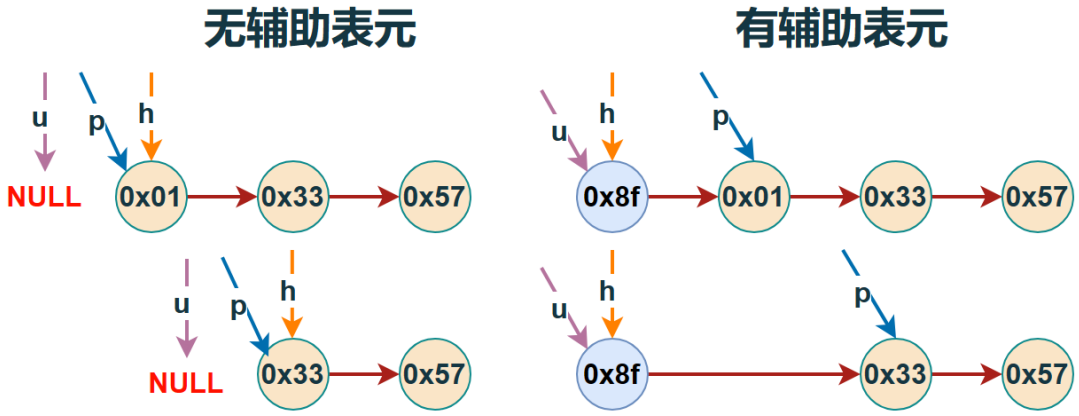
1. 字符串原文不需要用链表进行存储，使用数组即可；仅密文需要用链表存储。
2. 加密过程可概括为：构造空链表，连接链表首尾构成一个环，将原文字符逐个插入密文

链表的相应位置。

3. 寻找密文插入位置的方式：对于链表中还没插入字符的表元，可以将 data 初始化为 0，或者在结构定义中添加一个 flag 变量，用于表示表元是否已插入了字符。在插入字符的循环中，遇到未插入字符的表元则将计数器 i 加 1，直到 i 等于 key 便插入字符，并将计数器归零。重复以上过程直到原文中的所有字符都被插入了密文链表。

14. 用带辅助表元的有序整数链表表示整数集合，分别编写已知两个集合求集合和（$S_1=S_1 \cup S_2$）、集合差（$S_1=S_1-S_2$）、集合交（$S_1=S_1 \cap S_2$）的函数。运算结果在链表 S_1。

设 $S_1=\{2,3,5,6\}$，$S_2=\{3,4,6,8\}$，则有，集合和 $S_1 \cup S_2=\{2,3,4,5,6,8\}$，集合差 $S_1-S_2=\{2,5\}$，集合交 $S_1 \cap S_2=\{3,6\}$。




```c
#include <stdio.h>
#include <stdlib.h>

struct intNode
{
  int data;
  struct intNode *next;
};

struct intNode *tailInsert(int n)
{
  int i;
  struct intNode *head = (struct intNode *)malloc(sizeof(struct
intNode));
  struct intNode *h = NULL;
  struct intNode *tail;
  struct intNode *p;

  for (i = 0; i < n; ++i)
  {
    p = (struct intNode *)malloc(sizeof(struct intNode));
    scanf("%d", &(p->data));
```

```
    if (h == NULL)
      h = tail = p;
    else
    {
      tail->next = p;
      tail = p;
    }
  }
  tail->next = NULL;

  head->next = h;

  return head;
}

void print(struct intNode *h)
{
  struct intNode *p = h->next;
  for (; p; p = p->next)
  {
    printf("%d ", p->data);
  }
  printf("\n");
}

void free_data(struct intNode *h)
{
  struct intNode *p = h->next;
  struct intNode *u;

  for (; p;)
  {
    u = p->next;
    free(p);
    p = u;
  }
}

void and_set(struct intNode *h1, struct intNode *h2)
{
  struct intNode *u1, *w1, *u2;

  w1 = h1;
```

```
  u1 = h1->next;
  u2 = h2->next;
  while (u1 && u2)
  {
    if (u1->data < u2->data)
    { // u1 节点小于 u2 节点，说明 u1 在链表 2 中不存在，删除
      w1->next = u1->next;
      free(u1);
      u1 = w1->next;
    }
    else if (u1->data > u2->data)
    { // u1 大于 u2，无法判断是否为交，向后挪动 u2
      u2 = u2->next;
    }
    else
    { // u1 等于 u2，保留 u1,向后挪动
      w1 = u1;
      u1 = u1->next;
      u2 = u2->next;
    }
  }
  while (u1)
  { // 将链表 1 剩余的元素移除
    w1->next = u1->next;
    free(u1);
    u1 = w1->next;
  }
}

void or_set(struct intNode *h1, struct intNode *h2)
{
  struct intNode *u1, *w1, *u2, *p;

  w1 = h1;
  u1 = h1->next;
  u2 = h2->next;
  while (u1 && u2)
  {
    if (u1->data > u2->data)
    { // u1 大于 u2，说明 u2 在链表 1 中不存在，需要添加到链表 1，然后向后挪动 u2
      p = (struct intNode *)malloc(sizeof(struct intNode));
      p->data = u2->data;
      p->next = u1;
      w1->next = p;
```

```
      w1 = p;
      u2 = u2->next;
    }
    else if (u1->data < u2->data)
    { // 无法判断是否将 u2 添加到链表 1 中，向后挪动 u1
      w1 = u1;
      u1 = u1->next;
    }
    else
    { // u1->data 等于 u2->data，不需要将 u2 添加到链表 1，向后挪动 u1 u2
      w1 = u1;
      u1 = u1->next;
      u2 = u2->next;
    }
  }
  while (u2)
  { // 将链表 2 剩余的元素添加到链表 1
    p = (struct intNode *)malloc(sizeof(struct intNode));
    p->data = u2->data;
    p->next = NULL;
    w1->next = p; // w1 此时代表链表 1 的 tail
    w1 = p;
    u2 = u2->next;
  }
}

void diff_set(struct intNode *h1, struct intNode *h2)
{
  struct intNode *u1, *w1, *u2;

  w1 = h1;
  u1 = h1->next;
  u2 = h2->next;
  while (u1 && u2)
  {
    if (u1->data > u2->data)
    { // u2 在链表 1 中不存在，向后挪动 u2
      u2 = u2->next;
    }
    else if (u1->data < u2->data)
    { // 无法判断 u2 是否存在链表 1，向后挪动 u1
      w1 = u1;
      u1 = u1->next;
    }
```

```
    else
    { // u1 与 u2 值相等，从链表 1 中删除
      w1->next = u1->next;
      free(u1);
      u1 = w1->next;
      u2 = u2->next;
    }
  }
  // 无需再判断 u1 或 u2 是否为空
}

int main()
{
  struct intNode *h1, *h2;
  // 假设有序链表按照从小到大的顺序排列
  h1 = tailInsert(4);
  print(h1);

  h2 = tailInsert(4);
  print(h2);

  // 由于集合操作会改变 h1，所以每次程序只执行一个集合操作
  //  and_set(h1, h2);
  //  or_set(h1, h2);
  diff_set(h1, h2);
  print(h1);

  free_data(h1);
  free_data(h2);

  return 0;
}
```

两个有序链表的集合操作、合并操作，其套路都比较固定，就是以链表 1 为基础，将需要添加或删除的表元都放在链表 1 上进行，而尽量不去改变链表 2 的表元。循环遍历时的循环条件为 u1&&u2，根据 u1->data 与 u2->data 的大小关系分三种情况来讨论具体如何操作。对于从小到大排列的有序链表，u1->data > u2->data 说明 u2 表元在链表 1 中不存在，u1->data < u2->data 说明 u1 表元在链表 2 中不存在，但还无法判断 u2 在链表 1 中是否存在。

辅助表元的作用是确保链表的头表元是一个固定的表元，因此不必在代码中专门针对当前指针是否为头表元来添加额外的逻辑，可以简化代码。并且由于头表元不变，所以无需返回新的头表元地址，返回值为 void 即可。

第十四次作业解析

4. 编写将两个有序整数文件合并复制一个有序整数文件的程序，假定整数文件中的整数是从小到大排列的。要求新文件中的整数也从小到大排列，并且互不相同。

```c
#include <stdio.h>
#include <stdlib.h>

struct intNode
{
    int data;
    struct intNode *next;
};

int main(int argc, char **argv)
{
    FILE *fpr1, *fpr2, *fpw;
    int t;

    fpr1 = fopen(argv[1], "r");
    fpr2 = fopen(argv[2], "r");
    fpw = fopen(argv[3], "w"); // 代表写入结果的文件

    if (fpr1 == NULL || fpr2 == NULL || fpw == NULL) // 有任意文件打开失败都结束程序运行
        return -1;

    struct intNode *h1 = NULL, *h2 = NULL, *h = NULL, *v, *tail1, *tail2, *p, *u, *w;

    // 一边读入文件数据一边构建链表
    // EOF 代表 end of file, 若 fscanf 返回 EOF 则代表文件文本已全部读入
    while (fscanf(fpr1, "%d", &t) != EOF)
    {
        p = (struct intNode *)malloc(sizeof(struct intNode));
        p->data = t;
        if (h1 == NULL)
            h1 = tail1 = p;
        else
            tail1 = tail1->next = p;
    }
    p->next = NULL;
```

```
    while (fscanf(fpr2, "%d", &t) != EOF)
    {
        fscanf(fpr2, "%d", &t);
        p = (struct intNode *)malloc(sizeof(struct intNode));
        p->data = t;
        if (h2 == NULL)
            h2 = tail2 = p;
        else
            tail2 = tail2->next = p;
    }
    p->next = NULL;

    // 排序合并部分则按照常规的链表合并算法
    p = h1;
    while (p)
    {
        u = h2;
        v = p->next;
        while (u && u->data < p->data)
        {
            w = u;
            u = u->next;
        }

        if (u != NULL && u->data != p->data)
        {
            if (u == h2)
            {
                p->next = h2;
                h2 = p;
            }
            else
            {
                w->next = p;
                p->next = u;
            }
        }
        p = v;
    }

    p = h2;
    // 写入 fpw 指向的文件
    while (p)
    {
```

```
        fprintf(fpw, "%d ", p->data);
        p = p->next;
    }
    fclose(fpr1);
    fclose(fpr2);
    fclose(fpw);
    return 0;
}
```

对于文件处理相关的题目，建议将代码结构分为文件的数据读入（统一读入数组或者链表）、数据的处理（将结果统一存入数组和链表）、处理结果写入文件这三部分来编写，这样代码就不容易混乱。并且“数据的处理”部分可以直接复用以前学过的代码。

7. 编写程序检索指定的整数文件，统计文件中各个不同整数在文件中的出现次数。

```
#include <stdio.h>
#include <stdlib.h>

struct intNode
{
   int data;
   int cnt;
   struct intNode *next;
};

int main(int argc, char **argv)
{
   char filename[50];
   printf("Input file name:");
   scanf("%s", filename);
   FILE *fp = fopen(filename, "r");
   if (fp == NULL)
      return -1;

   int t;
   struct intNode *h = NULL, *tail, *p, *u, *v;

   // 读入文件里的每个数字，并依次存入链表(也可以是数组)
   while (fscanf(fp, "%d", &t) != EOF)
   {
      p = (struct intNode *)malloc(sizeof(struct intNode));
      p->data = t;
      p->cnt = 1;
```

```
        if (h == NULL)
            h = tail = p;
        else
            tail = tail->next = p;
    }

    tail->next = NULL;

    // 遍历链表，找出 data 相同的节点
    for (p = h; p; p = p->next)
    {
        // 从 p->next 开始向后遍历节点,u 指向 v 的上一个表元
        u = p;
        v = p->next;
        while (v)
        {
            // data 相同，则 p->cnt 加 1，然后从链表中移除节点 v
            if (v->data == p->data)
            {
                p->cnt++;
                u->next = v->next;
                free(v);
                v = u->next;
            }
            else
            {
                u = v;
                v = v->next;
            }
        }
    }

    for (p = h; p; p = p->next)
    {
        printf("%d: %d\n", p->data, p->cnt);
    }

    fclose(fp);

    return 0;
}
```

此题用数组也可以完成，利用数组下标来统计出现的数字，即 a[出现的数字]=数字出现的次数。但是数组需要定义其大小，并且文件中的数字大小并不能提前知道，所以不一定能保证

数组的大小足够大，因此采用链表是最合适的做法。

9. 设有一个整数文件，每行有若干个整数，要求编写程序，求文件中各行整数之和并输出到另一个文件。假定整数文件中每行的整数个数不定，每行最后一个整数之后可以有多余的空格符，也可能直接以换行符结束。

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv)
{
  FILE *fp = fopen(argv[1], "r");
  FILE *fpw = fopen(argv[2], "w");

  if (fp == NULL || fpw == NULL)
    return -1;

  char line[256];
  int sum;
  char *s;
  int i = 0;

  // 逐行读入文本进行处理，当读完所有文本后，fgets 返回 0，循环结束
  while (fgets(line, 256, fp))
  {
    // 遇到空格或者回车即对字符串进行分割
    // 第一次调用 strtok 来分割字符串 line 时，strtok 的第一个参数是 line
    s = strtok(line, " \n");
    sum = atoi(s);                  // atoi 将字符串文本对应的值转为 int 类型的值
    while (s = strtok(NULL, " \n")) // 第二次及之后调用 strtok 时，strtok 第一个参数为 NULL
      sum += atoi(s);               // 累加

    fprintf(fpw, "%d\n", sum); // 将结果输出到文件
  }

  fclose(fp);
  fclose(fpw);

  return 0;
}
```

此题的 `while (fgets(line, 256, fp))` 循环是逐行处理文本的套路，需要掌握。