图文作业6

3. 五只猴子采了一堆桃子，它们约定次日早晨起来再分。半夜里，一只猴子偷偷起来把桃子平均分成五堆后，发现还多一个，于是吃了这个桃子，拿走了其中一堆；第二只猴子醒来，又把桃子均分成五堆后，还是多了一个，它也吃了这个桃子，拿走了其中一堆；第三、第四、第五只猴子都依次如此做了。设计程序，求原先这堆桃子至少有多少个？最后剩多少个桃子？（已知 int 范围内有解。）

假设一开始有x_0个桃子：

$x_1=4/5(x_0-1)$

$x_2=4/5(x_1-1)$

…

$x_5=4/5(x_4-1)$

$x_4=5/4x_5+1$

$x_3=5/4x_4+1$

…

$x_0=5/4x_1+1$

```
int x5 = 4;
while (true) {
    int x = x5;
    int i = 0;
    for (i = 0; i < 5; i++) {
        if (x % 4 != 0)
            break;
        x = x / 4 * 5 + 1;
    }
    if (i == 5) {
        cout << x << endl;
        return 0;
    }
    x5 += 4;
}
```

图文作业7

3. 不引入新的数组，实现数组前 x 个元素与后 y 个元素交换位置的函数，其中 x+y 等于数组的长度。例如，设 a 中的数据为 1 2 3 4 5 6 7，设 x 为 3，y 为 4，函数执行后，a 中的数据为 4 5 6 7 1 2 3。又如，设 x 为 1，y 为 6，则函数执行后，a 中的数据为 2 3 4 5 6 7 1。函数原型为：void ArrSwap(int a[], int x, int y);

```
void ArrSwap(int a[], int x, int y) {
    int len = x + y;
    for (int i = 0; i < x; i++) { //循环左移x次
        int temp = a[0];
        for (int j = 1; j < len; j++) {
            a[j - 1] = a[j];
        }
        a[len - 1] = temp;
    }
}
```

图文作业8

1.（筛法求素数）首先假设 2 ~ n 之间所有的数从小到大排列在筛子中，然后从 2 开始依次将每个数的倍数从筛子中筛掉，最终输出筛子中剩下的数（均为素数）。

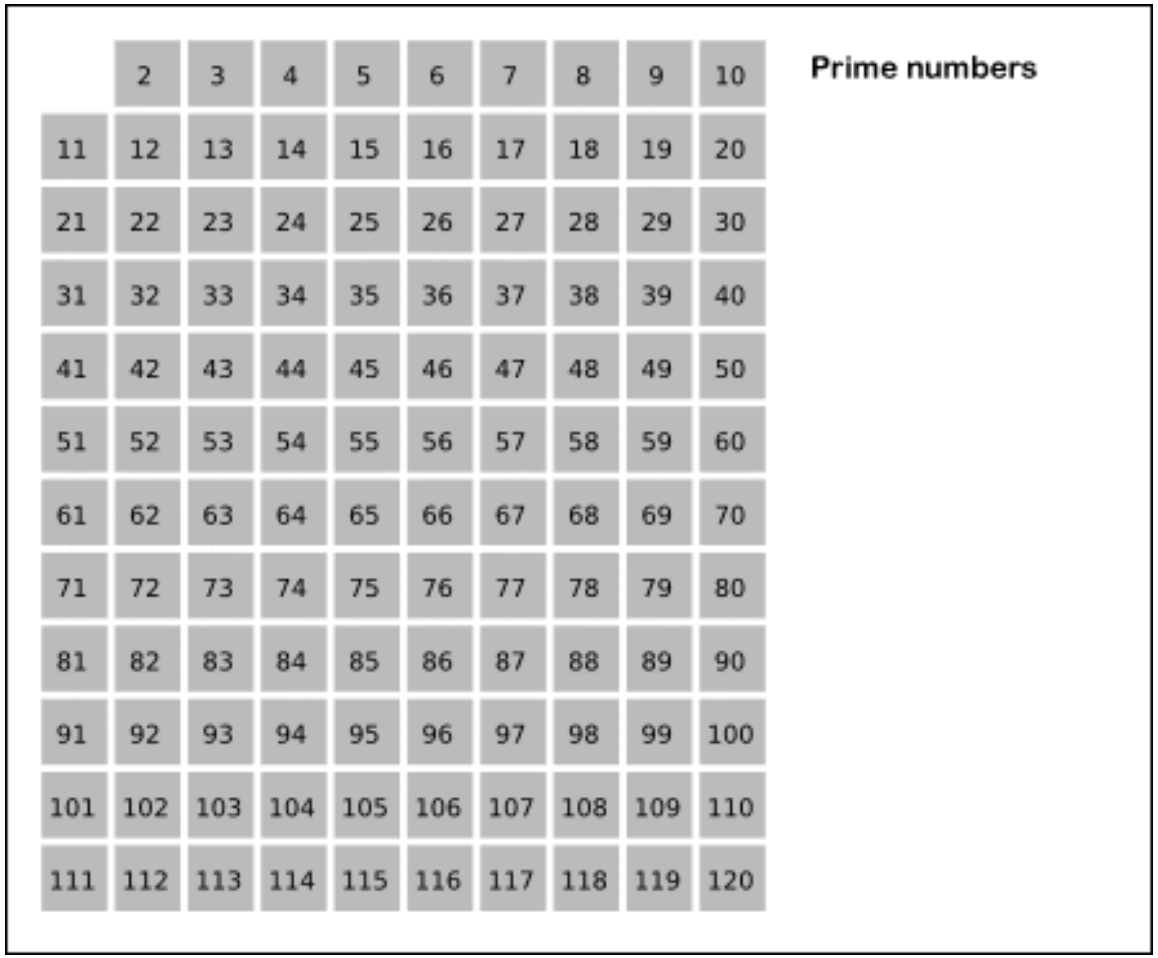
埃拉托斯特尼筛法通过不断地标记当前素数的所有倍数为合数，从而取得最小的未标记整数为下一个素数。不过，在实际使用此筛法寻找一个范围内的素数时，不需要检查范围内所有整数，也不需要对每个素数都标记其所有的倍数。

1. 寻找N以内的素数时，若找到了一个大于$\sqrt{N}$的素数，则剩余的所有尚未标记的数也都是素数。

   证明：若这些尚未标记的数中有任意一个为合数，设之为m，则m必定是除1与自身以外的两个因数的乘积。但既然m尚未被标记，则所有小于等于$\sqrt{N}$的数均不是m的因数。故这两个因数必然都大于$\sqrt{N}$，则m不可能在N以内[3]:103–104[4]:4。

2. 标记某一素数p的倍数时，不需要每次皆从$2p, 3p, \ldots$开始，而可直接从p^2开始标记。

   证明：所有较p^2更小的p的倍数必然拥有一个更小的素数为其因数，故在标记之前的素数的倍数时它们已经被标记过了[5]。



https://zh.wikipedia.org/wiki/%E5%9F%83%E6%8B%89%E6%89%98%E6%96%AF%E7%89%B9%E5%B0%BC%E7%AD%9B%E6%B3%95

图文作业8

1.（筛法求素数）首先假设 2 ~ n 之间所有的数从小到大排列在筛子中，然后从 2 开始依次将每个数的倍数从筛子中筛掉，最终输出筛子中剩下的数（均为素数）。

```
int eratosthenes(int n) {
    int p = 0;
    for (int i = 0; i <= n; i++) {
        is_prime[i] = true;
    }
    is_prime[0] = is_prime[1] = 0;
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) {
            prime[p++] = i;
            if (1ll * i * i <= n) {
                for (int j = i * i; j <= n;
j += i) {
                    is_prime[j] = 0;
                }
            }
        }
    }
    return p;
}
```

图文作业8

2.（前置和与后置和）假设有一个数组x，它有n个元素，每一个元素都大于0。称x[0]+x[1]+…+x[i]为前置和，而x[j]+x[j+1]+…+x[n-1]为后置和。编程计算x中有多少组相同的前置和与后置和。例如，如果x的元素为3、6、2、1、4、5、2，则x的前置和有：3、9、11、12、16、21、23，后置和有：2、7、1、12、14、20、23；11、12、23这3对就是值相同的前置和与后置和，因为：11=3+6+2（前置和）=2+5+4（后置和），12=3+6+2+1（前置和）=2+5+4+1（后置和），23是整个数组元素的和。

图文作业8

2. （前置和与后置和)

```
int sum1[n], sum2[n];
for (int i = 0; i < n; i++) {
    int rest1 = 0;
    int rest2 = 0;
    for (int j = 0; j <= i; j++) {
        rest1 += x[j];
        rest2 += x[n-j-1];
    }
    sum1[i] = rest1;
    sum2[i] = rest2;
}
```

```
int num = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {
        if (sum1[i] == sum2[j]) {
            num++;
            break;
        }
    }
}
```

图文作业9

动手实践，略。

图文作业10

1. 不使用库函数，编写 StrLen、StrCat、StrNcat、StrCmp 及 StrNcmp 函数。

请参考课件，略。

图文作业10

2. 用一个函数计算 main 函数中输入的一行字符中阿拉伯数字及大小写英文字母的个数，并用指针型参数将两类字符的个数传递给 main 函数输出。

```
char ch;
int n = 0;
while ((ch = getchar()) != '\n') {
    if ((ch >= '0' && ch <= '9') ||
        (ch >= 'a' && ch <= 'z') ||
        (ch >= 'A' && ch <= 'Z'))
        n++;
}
cout << n << endl;
```

图文作业11

2. 设计C/C++程序完成下列功能：设有 M个候选人，N 个选举人，每个选举人输入一个候选人姓名，最后输出各人得票数。

```
const int M = 3;
const int S = 20;
typedef struct
{
        char name[S];          //char name;?
        int count;            //double count;?

}Person;
Person leader[M] = {"Tom", 0, "Jerry", 0, "Tuffy", 0};
```

```
const int N = 10;
int i;
char name[S];
for(i=1; i <= N; ++i)
{
        cin.getline(name, S);                          //投票，输入选谁
        for(int j=0; j < M; ++j)
                if(strcmp(name, leader[j].name) == 0)
                        leader[j].count++;             //计票
}
cout << endl;
for(i=0; i < M; ++i)
        cout << leader[i].name << leader[i].count;     //唱票
```

```
cin >> m;
cin >> n;
getchar();
```

```
scanf("%d%d\n", &m, &n);
```

链表解法

```
struct Node
{
	char name[20];
	int num;
	Node *next;
};
```

```
Node *ListCreate_Tail(int m)
{
	Node *head = NULL, *tail = NULL;
	for(int i = 0; i <= m; ++i)
	{
		Node *p = (Node *)malloc(sizeof(Node));
		scanf("%s", p -> name);
		p -> num = 0;
		p -> next = NULL;
……
```

链表解法

```
struct Node
{
	char name[20];
	int num;
	Node *next;
};
```

```
Node *ListCreate_Tail(int m)
{
	Node *head = NULL, *tail
	for(int i = 0; i <= m; ++
	{
		Node *p = (Node *)r
		scanf("%s", p -> na
		p -> num = 0;
		p -> next = NULL;
……
```

```
int m, n;
cin >> m;
cin >> n;
Node *p = ListCreate_Tail(m);
for(int i = 0; i < n; ++i)
{	char vote[20];
	scanf("%s", vote);
	Node *q = NULL;
	for(q = p; q != NULL; q = q -> next)
	{
		if(strcmp(q -> name, vote) == 0)
		q -> num ++;
	}
}
p = p -> next;//因为第一个节点是空节点，姓名为'\n'
while(p)
{	cout << p -> num;
	p = p -> next;
}
```

图文作业12

1.（单向链表中的环）判断一个单向链表中是否有环（即最后一个节点的next指针是否指向了链表中的某个节点），并对无环的非空单向链表建立一个环（从链表最后一个节点指向第M个节点，其中，M可以是1、2、3、...），操作成功，返回true，否则（链表空），返回false。函数原型分别为：bool HasLoop(Node *head);及bool CreateLoop(Node *head, int m);

图文作业12

```
bool CreateLoop(Node *head, int m) {
    Node *temp = head;
    for (int i = 0; i < m-1 && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL)
        return false;

    Node *tail = head;
    while (tail->next != NULL) {
        tail = tail->next;
    }
    tail->next = temp;
}
```

图文作业12

```
bool HasLoop(Node *head) {
    if (head == NULL || head->next == NULL)
        return false;
    Node *fast = head;
    Node *slow = head;
    while (fast != NULL && fast->next != NULL) {
        fast = fast->next->next;
        slow = slow->next;
        if (fast == slow)
            return true;
    }
    return false;
}
```

图文作业12

2. 用链表实现约瑟夫斯问题（参见课件“5数组”例 5.7）的求解。

```
Node *JosephCircle(Node *head, int k) {
    Node *temp = head, *prev = NULL;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head;
    temp = head;
    while (temp->next != temp) {
        for (int i = 1; i < k; i++) {
            prev = temp;
            temp = temp->next;
        }
        prev->next = temp->next;
        delete temp;
        temp = prev->next;
    }
    temp->next = NULL;
    return temp;
}
```