2005年复试上机题

要求：

把一个数表示成若干个素数的和.

程序：

```
#include <stdio.h>

bool is_prime(int num)
{
    if (num < 2)
        return false;
    for (int i = 2; i * i <= num; i++)
        if (num % i == 0)
            return false;
    return true;
}

void split(int num)
{
    int i;

    if (num < 2)
        return;
    if (is_prime(num))
    {
        printf("%d ", num);
        return;
    }
    for (i = num; i > 1; --i)
    {
```

```
        if (is_prime(i) && num - i > 1)
        {
            printf("%d ", i);
            split(num - i);
            return;
        }
    }
}

int main(void)
{
    int i;

    while (scanf("%d", &i))
    {
        split(i);
        printf("\n");
    }
    getchar();
}
```

要求：

统计篇文章中各英文字母的个数，并排序.

程序：

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

typedef struct
{
    char c;
    int n;
} Letter;

void swap(Letter* a, Letter* b)
{
    Letter temp = *a;

    *a = *b;
    *b = temp;
}

int compare(Letter* a, Letter* b)
{
    if (a->n < b->n)
        return -1;
    else if (a->n > b->n)
        return 1;
    else
        return 0;

}
```

```
void selection_sort(Letter* ptr, int count)
{
    for (int i = 0; i < count; i++)
        for (int j = i + 1; j < count; j++)
            if (compare(&ptr[i], &ptr[j]) > 0)
                swap(&ptr[i], &ptr[j]);
}

bool is_upper(char c) { return c >= 'A' && c <= 'Z'; }
bool is_lower(char c) { return c >= 'a' && c <= 'z'; }

int main(void)
{
    FILE* fp = fopen("address.txt", "r");
    Letter letter[26];
    char c;

    if(!fp)
    {
        printf("File opening failed");
        return EXIT_FAILURE;
    }
    for (int i = 0; i < 26; i++)
    {
        letter[i].c = i + 'a';
        letter[i].n = 0;
    }
    while ((c = fgetc(fp)) != EOF)
    {
        if (is_upper(c))
            letter[c -'A'].n++;
        else if (is_lower(c))
            letter[c -'a'].n++;
    }
    selection_sort(letter, 26);
    for (int i = 0; i < 26; i++)
        printf("%c: %d\n", letter[i].c, letter[i].n);
    fclose(fp);
}
```

`2006` 年复试上机题

要求:

找出 `100` 到 `1000` 内的不含 `9` 的素数，存到 `result` 文件中.

程序:

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

```
bool is_prime(int num)
{
    if (num < 2)
        return false;
    for (int i = 2; i * i <= num; i++)
        if (num % i == 0)
            return false;
    return true;
}

bool has9(int num)
{
    if (num < 0)
        return false;
    while (num > 0)
    {
        if (num % 10 == 9)
            return true;
        num /= 10;
    }
    return false;
}

int main()
{
    FILE* fp = fopen("result.txt", "w");

    for (int i = 100; i < 1000; i++)
        if (is_prime(i) && !has9(i))
            fprintf(fp, "%d\n", i);
    fclose(fp);
}
```

`2007`年复试上机题

要求：

把`10`到`1000`之间满足以下两个条件的数，存到`result.txt`文件中.

- 是素数.
- 它的反数也是素数，如：`123`的反数是`321`.

程序：

```
#include <stdio.h>

bool is_prime(int num)
{
    if (num < 2)
        return false;
    for (int i = 2; i * i <= num; i++)
        if (num % i == 0)

            return false;
```

```
    return true;
}

int reverse(int num)
{
    int rev = 0;

    while (num > 0)
    {
        rev = rev * 10 + num % 10;
        num /= 10;
    }
    return rev;
}

int main()
{
    FILE* fp = fopen("result.txt", "w");

    for (int i = 100; i < 1000; i++)
        if (is_prime(i) && is_prime(reverse(i)))
            fprintf(fp, "%d %d\n", i, reverse(i));
    fclose(fp);
}
```

`2008` 年复试上机题

要求：

- 用 `IE` 从 `FTP` 上下载 `org.dat`，并保存在 `D` 盘的根目录中.
- 此文件中按文本方式存放了一段其他文章，其中有若干长度小于 `15` 的英文单词，单词之间用空格分开，无其他符号.
- 顺序读取这段文章的不同的单词(大小写敏感)，同时在读取的过程中排除所有的单词 `THE` 以及变形，即这些单词不能出现在读取的结果中.
- 将读取的所有单词的首字母转大写后，输出 `D` 根目录下 `new.txt`，每个单词一行.

程序：

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

bool is_the(char word[15])
{
    char the[] = "the";

    if (strlen(word) != strlen(the))
        return false;
    for (int i = 0; i < strlen(word); i++)
        word[i] |= 0x20;
    return strcmp(word, the) == 0;
}
```

```
int main()
{
    FILE* fporg = fopen("org.dat", "r");
    FILE* fpnew = fopen("new.txt", "w");
    char word[15];

    if(!fporg || !fpnew)
    {
        printf("File opening failed");
        return EXIT_FAILURE;
    }
    printf("org.dat:\n");
    while (fscanf(fporg, "%s", word) != EOF)
    {
        printf("%s ", word);
        if (!is_the(word))
        {
            word[0] = (word[0] | 0x20) - 0x20;
            fprintf(fpnew, "%s\n", word);
        }
    }
    printf("\n");
    fclose(fporg);
    fclose(fpnew);

    fpnew = fopen("new.txt", "r");
    printf("new.txt:\n");
    while (fscanf(fporg, "%s", word) != EOF)
        printf("%s ", word);
    printf("\n");
    fclose(fpnew);
    return 0;
}
```

输出:

```
org.dat:
The constructor is used to initialize the object The destructor is used to delete the Object the
calling seqence of constructor is opposite to the calling sequence of destructor
new.txt:
Constructor Is Used To Initialize Object Destructor Is Used To Delete Object Calling Seqence Of
Constructor Is Opposite To Calling Sequence Of Destructor
```

`2009` 年复试上机题

要求:

- 用 `IE` 浏览器从 `FTP` 上下载 `org.dat`，并保存在 `D` 盘的根目录下.
- 此文件中按文本方式存放了一段其他文章，其中有若干长度小于 `15` 的十进制或八进制数字，数字之间用 `,` 分开，数字内部存在且仅存在空格.
- 八进制数以起始位 `0` 作为标示与十进制数区分.

- 顺序读取这些数字将他们转变为十进制数后按从大到小的顺序排序后，输出到`D`盘根目录下`new.txt`，每个数字一行.

  `eg`：`_235_,34__2,_043_1_,1_3`，分别是：十进制`235`，十进制`342`，八进制`431`，十进制`13`，`_`代表空格.

程序：

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

void swap(int* a, int* b)
{
    int temp = *a;

    *a = *b;
    *b = temp;
}

int compare(int* a, int* b)
{
    if (a < b)
        return -1;
    else if (a > b)
        return 1;
    else
        return 0;
}

void selection_sort(int* ptr, int count)
{
    for (int i = 0; i < count; i++)
        for (int count = i + 1; count < count; count++)
            if (compare(&ptr[i], &ptr[count]) > 0)
                swap(&ptr[i], &ptr[count]);
}

bool is_num(char c) { return c >= '0' && c <= '9'; }
bool valid(char c) { return is_num(c) || c == ' '; }

int main()
{
    FILE* fporg = fopen("org.dat", "r");
    FILE* fpnew = fopen("new.txt", "w");
    int i;
    int count = 0;
    int arr[128];
    char c;
    char num[15];

    printf("org.dat:\n");

    while ((c = fgetc(fporg)) != EOF)
```

```
    {
        if (is_num(c))
        {
            i = 0;
            num[i++] = c;
            while ((c = fgetc(fporg)) != EOF && valid(c))
                if (is_num(c)) num[i++] = c;
            num[i] = '\0';
            printf("%s\n", num);
            arr[count++] = num[0] == '0' ?
                           strtol(num, NULL, 8) :
                           strtol(num, NULL, 10);
        }
    }
    selection_sort(arr, count);
    for (i = 0; i < count; i++)
        fprintf(fpnew, "%d\n", arr[i]);
    fclose(fporg);
    fclose(fpnew);

    fpnew = fopen("new.txt", "r");
    printf("new.txt:\n");
    while (fgets(num, sizeof(num), fpnew))
        printf("%s", num);
    fclose(fpnew);
}
```

输出:

```
org.dat:
235
342
0431
13
new.txt:
235
342
281
13
```

`2010` 年复试上机题

要求:

- 从 `FTP` 上下载 `make.exe` 和 `org.dat`，运行 `make.exe` 输入准考证后三位生成 `data.txt`，文件为二进制编码.
- `data.txt` 内存有 `2048` 个整数，其中前 `n` 个为非 `0` 数，后 `2048-n` 个数为 `0`，将其读入数组，计算非零数的个数 `n`.
- 选出 `n` 个数中的最大数和最小数.
- 选出 `n` 个数中最大素数.

- 将`n`个数从大到小排序，并平均分成三段(若`n`非`3`的整数倍，则不考虑最后的`1-2`个数)，选出中间段的最大数和最小数.

程序：

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

bool is_prime(int num)
{
    if (num < 2)
        return false;
    for (int i = 2; i * i <= num; i++)
        if (num % i == 0)
            return false;
    return true;
}

void swap(int* a, int* b)
{
    int temp = *a;

    *a = *b;
    *b = temp;
}

int compare(int* a, int* b)
{
    if (a < b)
        return -1;
    else if (a > b)
        return 1;
    else
        return 0;
}

void selection_sort(int* ptr, int count)
{
    for (int i = 0; i < count; i++)
        for (int count = i + 1; count < count; count++)
            if (compare(&ptr[i], &ptr[count]) > 0)
                swap(&ptr[i], &ptr[count]);
}

int main(void)
{
    FILE* fp;
    int num[2048];
    int min = RAND_MAX;

    int max = 0;
```

```
    int max_prime = 0;
    int n, i;

    fp = fopen("data.txt", "wb");
    if(!fp)
    {
        printf("File opening failed");
        return EXIT_FAILURE;
    }
    srand(time(NULL));
    for (i = 0; i < 1000; i++)
        num[i] = rand() % 4096;
    for (; i < 2048; i++)
        num[i] = 0;
    fwrite(num, sizeof(int), 2048, fp);
    fclose(fp);

    fp = fopen("data.txt", "rb");
    if(!fp)
    {
        printf("File opening failed");
        return EXIT_FAILURE;
    }
    fread(num, sizeof(int), 2048, fp);
    for (n = 0; n < 2048; n++)
        if (num[n] == 0)
            break;
    printf("n = %d\n", n);
    for (i = 0; i < n; i++)
    {
        if (min > num[i]) min = num[i];
        if (max < num[i]) max = num[i];
        if (is_prime(num[i]) && num[i] > max_prime) max_prime = num[i];
    }
    printf("min = %d\nmax = %d\nmax_prime = %d\n", min, max, max_prime);
    selection_sort(num, n);
    printf("min in mid_seg = %d\nmax in mid_seg = %d\n", num[n / 3 * 2 - 1], num[n / 3]);

    fclose(fp);
}
```

输出：

```
n = 1000
min = 1
max = 4095
max_prime = 4093
min in mid_seg = 3630
max in mid_seg = 1863
```

`2011` 年复试上机题

要求：

输出`1000-9999`中满足以下条件的所有数：

- 该数是素数.
- 十位数和个位数组成的数是素数，百位数和个位数组成的数是素数.
- 个位数和百位数组成的数是素数，个位数和十位数组成的数是素数. 比如`1991`，个位和十位组成的数就是`19`.

程序：

```
#include <stdio.h>

bool is_prime(int num)
{
    if (num < 2)
        return false;
    for (int i = 2; i * i <= num; i++)
        if (num % i == 0)
            return false;
    return true;
}

int main(void)
{
    for (int i = 1000; i <= 9999; i++)
    {
        int g = i % 10;
        int s = (i / 10) % 10;
        int b = (i / 100) % 10;

        if (is_prime(i) &&
            is_prime(i % 100) &&
            is_prime(g * 10 + b) &&
            is_prime(g * 10 + s) &&
            is_prime(b * 10 + g))
            printf("%d ", i);
    }
}
```

输出：

```
1117 1171 1997 2111 2113 2131 2137 2311 2371 2711 2713 2731 2917 3137 3331 3371 3779 3917 4111
4337 4397 4937 5113 5171 5197 5711 5779 6113 6131 6173 6197 6311 6317 6337 6397 6779 6917 6997
7331 7937 8111 8117 8171 8311 8317 8713 8731 8779 9137 9173 9311 9337 9371 9397
```

`2012`年复试上机题

要求：

- 从服务器上下载数据文件`org.dat`文件以二进制方式存放一系列整数，每个整数占`4`个字节. 从第一个整数开始，第一个整数和第二个整数构成一个坐标点，以此类推，数据文件中保存了许多坐标点数据.

- 规定处于第一象限的坐标点为有效点，请问数据文件中所有点的个数`n`为多少？有效点的个数`k`为多少？
- 每个有效点与坐标原点构成一个的矩形，请问`k`个有效点与坐标原点构成的`k`个矩形的最小公共区域面积为多少？
- 寻找有效点钟符合下列条件的点：以该点为坐标原点，其它有效点仍然是有效点即处于第一象限(不包括坐标轴上的点). 输出这些点.
- 对所有有效点进行分组，每个有效点有且只有属于一个分组，分组内的点符合下列规则：若对组内所有点的x坐标进行排序，点`p1(x1, y1)`在点`p2(x2, y2)`后面，即`x1>x2`那么`y1>y2`，请输出所有的分组.

程序：

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

typedef struct
{
    int x;
    int y;
} Coordinate;

void swap(Coordinate* a, Coordinate* b)
{
    Coordinate temp = *a;

    *a = *b;
    *b = temp;
}

int compare(const Coordinate* lhs, const Coordinate* rhs)
{
    if (lhs->x < rhs->x)
        return -1;
    if (lhs->x > rhs->x)
        return 1;
    else
        return 0;
}

void selection_sort(Coordinate* ptr, int begin, int end)
{
    for (int i = begin; i < end; i++)
        for (int j = i + 1; j < end; j++)
            if (compare(&ptr[i], &ptr[j]) > 0)
                swap(&ptr[i], &ptr[j]);
}

void display(Coordinate* ptr, int count)
{
    for (int i = 0; i < count; i++)

        printf("(%2d, %2d) ", ptr[i].x, ptr[i].y);
```

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

int main(void)
{
    FILE* fp;
    Coordinate* xy;
    int* num;
    int n, count;

    fp = fopen("org.dat", "wb");
    num = (int*) malloc(sizeof(int) * 128);
    srand(time(NULL));
    for (int i = 0; i < 128; i++)
        num[i] = rand() % 128 - 64;
    fwrite(num, sizeof(int), 128, fp);
    free(num);
    fclose(fp);

    fp = fopen("org.dat", "rb");
    fseek(fp, 0, SEEK_END);
    count = ftell(fp) / sizeof(int);
    n = count / 2;
    num = (int*) malloc(sizeof(int) * count);
    xy = (Coordinate*) malloc(sizeof(Coordinate) * n);
    rewind(fp);
    fread(num, sizeof(int), count, fp);
    fclose(fp);

    int k = 0;
    int minx = RAND_MAX;
    int miny = RAND_MAX;
    for (int i = 0; i < count; i += 2)
    {
        if (num[i] > 0 && num[i + 1] > 0)
        {
            xy[k].x = num[i];
            xy[k].y = num[i + 1];
            if (minx > xy[k].x)
                minx = xy[k].x;
            if (miny > xy[k].y)
                miny = xy[k].y;
            k++;
        }
    }
    selection_sort(xy, 0, k);
    printf("valid points:\n");
    display(xy, k);
    printf("n = %d k = %d\n", n, k);
    printf("min area = %d\n", minx * miny);

    int sorted = 0;

    while (sorted < k)
```

```
    {
        selection_sort(xy, sorted, k);
        printf("points grouped by (%2d, %2d): ", xy[sorted].x, xy[sorted].y);
        miny = xy[sorted++].y;
        for (int i = sorted; i < k; i++)
        {
            if (xy[i].y >= miny)
            {
                miny = xy[i].y;
                printf("(%d, %d) ", xy[i].x, xy[i].y);
                swap(&xy[i], &xy[sorted++]);
            }
        }
        printf("\n");
    }
}
```

输出：

```
valid points:
( 3,  8) (10, 62) (11, 40) (20,  6) (26,  7) (26, 20) (29, 31) (33, 10) (34, 37) (37, 33) (50,
55) (52, 59) (53, 49) (53,  8) (57, 16) (58, 20)
n = 64 k = 16
min area = 18
points grouped by ( 3,  8): (10, 62)
points grouped by (11, 40): (50, 55) (52, 59)
points grouped by (20,  6): (26, 20) (29, 31) (34, 37) (53, 49)
points grouped by (26,  7): (33, 10) (37, 33)
points grouped by (53,  8): (57, 16) (58, 20)
```

`2013` 年复试上机题

要求：

- **Introduction**

  The project will read flight data from an input file and flight path requests from another input file and output the required information.

- **Your Task**

  Your program should determine ***if a particular destination airport can be reached from a particular originating airport within a particular number of hops.*** A hop (leg of a flight) is a flight from one airport to another on the path between an originating and destination airports. For example, the flight plan from `PVG` to `PEK` might be `PVG -> CAN -> PEK`. So `PVG -> CAN` would be a hop and `CAN -> PEK` would be a hop.

- **Input Data Files**

  - Path Input File(`PathInput.txt`)

    This input file will consist of a number of single origination/destination airport pairs (direct flights). The first line of the file will contain an integer representing the total number of pairs in the rest of the file.

```
6
[PVG, CAN]
[CAN, PEK]
[PVG, CTU]
[CTU, DLC]
[DLC, HAK]
[HAK, LXA]
```

- Path Request File(`PathRequest.txt`)

  This input file will contain a sequence of pairs of origination/destination airports and a max number of hops. The first line of the file will contain an integer representing the number of pairs in the file.

  ```
  2
  [PVG, DLC, 2]
  [PVG, LXA, 2]
  ```

- Output File(`Output.txt`)

  For each pair in the Path Request File, your program should output the pair followed by `YES` or `NO` indicating that it is possible to get from the origination to destination airports within the max number of hops or it is not possible, respectively.

  ```
  [PVG, DLC, YES]
  [PVG, LXA, NO]
  ```

- **Assumptions you can make:**

  You may make the following simplifying assumptions in your project:

  - `C/C++` is allowed to be used.
  - All airport codes will be `3` letters and will be in all caps
  - Origination/destination pairs are unidirectional. To indicate that both directions of flight are possible, two entries would appear in the file. For example, `[PVG, PEK]` and `[PEK, PVG]` would have to be present in the file to indicate that one could fly from `Shanghai` to `Beijing` and from `Beijing` to `Shanghai`.

程序:

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

typedef struct Edge
{
    int index;
    char city[4];
    struct Edge* next;

} Edge;
```

```
typedef struct
{
    char city[4];
    Edge* first;
} Vertex;
typedef struct
{
    int size;
    int capacity;
    Vertex* vertices;
} Graph;

int find(Vertex* vertices, int count, char src[4])
{
    int i;

    for (i = 0; i < count; i++)
        if (strcmp(src, vertices[i].city) == 0)
            break;
    return i;
}

void add_edge(Graph* g, char src[4], char dst[4])
{
    int i, j;
    Edge* e;

    i = find(g->vertices, g->size, src);
    if (i == g->size)
    {
        strcpy(g->vertices[g->size].city, src);
        g->vertices[g->size].first = NULL;
        g->size++;
    }
    j = find(g->vertices, g->size, dst);
    if (j == g->size)
    {
        strcpy(g->vertices[g->size].city, dst);
        g->vertices[g->size].first = NULL;
        g->size++;
    }
    for (e = g->vertices[i].first; e != NULL; e = e->next)
        if (strcmp(e->city, dst) == 0)
            break;
    if (e == NULL)
    {
        e = (Edge*) malloc(sizeof(Edge));
        strcpy(e->city, dst);
        e->index = j;
        e->next = g->vertices[i].first;
        g->vertices[i].first = e;
    }

}
```

```
bool is_possible(Graph* g, bool* visited, char src[], char dst[], int step)
{
    int current;
    Edge* e;

    current = find(g->vertices, g->size, src);
    if (step == 0 || visited[current])
        return false;
    else
    {
        visited[current] = true;
        for (e = g->vertices[current].first; e != NULL; e = e->next)
        {
            if (strcmp(e->city, dst) == 0 && step == 1)
                return true;
            else if (is_possible(g, visited, e->city, dst, step - 1))
                return true;
        }
        visited[current] = false;
    }
    return false;
}

void destruct(Graph* g)
{
    for (int i = 0; i < g->size; i++)
    {
        while (g->vertices[i].first != NULL)
        {
            Edge* e = g->vertices[i].first->next;
            free(g->vertices[i].first);
            g->vertices[i].first = e;
        }
    }
    free(g->vertices);
}

void display(Graph* g)
{
    printf("adjacency list:\n");
    for (int i = 0; i < g->size; i++)
    {
        printf("%s: ", g->vertices[i].city);
        for (Edge* edge = g->vertices[i].first; edge != NULL; edge = edge->next)
            printf("%s ", edge->city);
        printf("\n");
    }
    printf("\n");
}

int main(void)

{
```

```
    FILE* fpi = fopen("PathInput.txt", "r");
    FILE* fpr = fopen("PathRequest.txt", "r");
    FILE* fpo = fopen("Output.txt", "w");
    char line[16];
    char src[4];
    char dst[4];
    bool* visited;
    int step;
    int num;
    Graph g;

    if (!fpi || !fpr || !fpo)
    {
        printf("failed to open files.\n");
        exit(-1);
    }
    memset(line, 0, sizeof(line));
    if (fgets(line, sizeof(line), fpi) != NULL)
    {
        g.size = 0;
        g.capacity = atoi(line) * 2;
        g.vertices = (Vertex*) malloc(g.capacity * sizeof(Vertex));
        memset(g.vertices, 0, g.capacity);
    }
    while (fgets(line, sizeof(line), fpi) != NULL)
    {
        memcpy(src, line + 1, 3);
        src[3] = '\0';
        memcpy(dst, line + 6, 3);
        dst[3] = '\0';
        add_edge(&g, src, dst);
    }
    display(&g);

    printf("flight info:\n");
    visited = (bool*) malloc(g.size);
    if (fgets(line, sizeof(line), fpr) != NULL)
        num = atoi(line);
    while (fgets(line, sizeof(line), fpr) != NULL)
    {
        memcpy(src, line + 1, 3);
        src[3] = '\0';
        memcpy(dst, line + 6, 3);
        dst[3] = '\0';
        step = line[11] - '0';
        memset(visited, false, g.size);
        if (is_possible(&g, visited, src, dst, step))
        {
            printf("[%s, %s, %d, YES]\n", src, dst, step);
            fprintf(fpo, "[%s, %s, YES]\n", src, dst);
        }
        else

        {
```

```
            printf("[%s, %s, %d, NO]\n", src, dst, step);
            fprintf(fpo, "[%s, %s, NO]\n", src, dst);
        }
    }

    destruct(&g);
    free(visited);
    fclose(fpi);
    fclose(fpr);
    fclose(fpo);
}
```

输出：

```
adjacency list:
PVG: CTU CAN
CAN: PEK
PEK:
CTU: DLC
DLC: HAK
HAK: LXA
LXA:

flight info:
[PVG, DLC, 2, YES]
[PVG, LXA, 2, NO]
```

`2014`年复试上机题

要求：

- 从网页上下载`input.dat`文件，里面是用二进制编写的，里面放了一堆`int`型的数，每个数占`4`个字节，每次读取两个，这两个数构成一个坐标.
- 规定处于第一象限的数是有效点(即`x>0, y>0`的坐标)，问这么多点中有效点有多少个?
- 现在用户从键盘输入一个坐标和一个数字`k`，设计算法输出`k`个离该坐标距离最近的点的坐标和每个坐标到该点的距离，写入到`output.txt`文件中.

程序：

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

typedef struct
{
    int x;
    int y;
} Coordinate;
```

```
Coordinate point;

double distance(const Coordinate* a, const Coordinate* b)
{
    int val = (a->x - b->x) * (a->x - b->x) +
              (a->y - b->y) * (a->y - b->y);
    return sqrt((double) val);
}

void swap(Coordinate* a, Coordinate* b)
{
    Coordinate temp = *a;

    *a = *b;
    *b = temp;
}

int compare(const Coordinate* a, const Coordinate* b)
{
    double dista = distance(a, &point);
    double distb = distance(b, &point);

    if (dista < distb)
        return -1;
    if (dista > distb)
        return 1;
    else
        return 0;
}

void selection_sort(Coordinate* ptr, int begin, int end)
{
    for (int i = begin; i < end; i++)
        for (int j = i + 1; j < end; j++)
            if (compare(&ptr[i], &ptr[j]) > 0)
                swap(&ptr[i], &ptr[j]);
}

void display(Coordinate* ptr, int count)
{
    for (int i = 0; i < count; i++)
        printf("(%2d, %2d) ", ptr[i].x, ptr[i].y);
    printf("\n");
}

int main(void)
{
    FILE* fp;
    Coordinate* xy;
    int* num;
    int count, coord_count;


    fp = fopen("org.dat", "wb");
```

```
    num = (int*) malloc(sizeof(int) * 128);
    srand(time(NULL));
    for (int i = 0; i < 128; i++)
        num[i] = rand() % 128 - 64;
    fwrite(num, sizeof(int), 128, fp);
    free(num);
    fclose(fp);

    fp = fopen("org.dat", "rb");
    fseek(fp, 0, SEEK_END);
    count = ftell(fp) / sizeof(int);
    coord_count = count / 2;
    num = (int*) malloc(sizeof(int) * count);
    xy = (Coordinate*) malloc(sizeof(Coordinate) * coord_count);
    rewind(fp);
    fread(num, sizeof(int), count, fp);
    fclose(fp);

    int valid = 0;
    fp = fopen("output.txt", "w");
    for (int i = 0; i < count; i += 2)
    {
        if (num[i] > 0 && num[i + 1] > 0)
        {
            xy[valid].x = num[i];
            xy[valid].y = num[i + 1];
            valid++;
        }
    }
    printf("valid points:\n");
    display(xy, valid);
    printf("total: %d\n", valid);

    int k;
    printf("coordinate: ");
    scanf("%d %d", &point.x, &point.y);
    printf("k = ");
    scanf("%d", &k);
    selection_sort(xy, 0, valid);
    printf("the nearest %d points to (%d, %d) are:\n", k, point.x, point.y);
    for (int i = 0; i < k; i++)
    {
        printf("(%2d, %2d) %7.2f\n", xy[i].x, xy[i].y, distance(&xy[i], &point));
        fprintf(fp, "(%2d, %2d) %7.2f\n", xy[i].x, xy[i].y, distance(&xy[i], &point));
    }
    fclose(fp);
}
```

输出:

```
valid points:
(49, 38) (59, 45) (53, 19) (34, 55) (23, 62) ( 2, 11) (14, 14) (55, 52) (62, 37) (46, 29) (19,
57) (12, 18) ( 4,  7)
total: 13
coordinate: 0 0
k = 5
the nearest 5 points to (0, 0) are:
( 4,  7)    8.06
( 2, 11)   11.18
(14, 14)   19.80
(12, 18)   21.63
(46, 29)   54.38
```

`2015`年复试上机题

要求：

- 从网页上下载`input.dat`文件，里面是用二进制编写的，里面放了一堆`int`型的数，每个数占`4`个字节，每次读取两个，这两个数构成一个坐标.
- 规定处于第一象限的数是有效点(即`x>0, y>0`的坐标)，问这么多点中有效点有多少个?
- 从键盘上输入`k`和`n`，从第一问中的有效点中找出距离小于`n`，距离小于`n`的点的个数要大于`k`，将它们以文本格式输出到文件中.

程序：

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

typedef struct
{
    int x;
    int y;
} Coordinate;

Coordinate point;

double distance(const Coordinate* a, const Coordinate* b)
{
    int val = (a->x - b->x) * (a->x - b->x) +
              (a->y - b->y) * (a->y - b->y);
    return sqrt((double) val);
}

void swap(Coordinate* a, Coordinate* b)
{
    Coordinate temp = *a;


    *a = *b;
```

```
    *b = temp;
}

int compare(const Coordinate* a, const Coordinate* b)
{
    double dista = distance(a, &point);
    double distb = distance(b, &point);

    if (dista < distb)
        return -1;
    if (dista > distb)
        return 1;
    else
        return 0;
}

void selection_sort(Coordinate* ptr, int begin, int end)
{
    for (int i = begin; i < end; i++)
        for (int j = i + 1; j < end; j++)
            if (compare(&ptr[i], &ptr[j]) > 0)
                swap(&ptr[i], &ptr[j]);
}

void display(Coordinate* ptr, int count)
{
    for (int i = 0; i < count; i++)
        printf("(%2d, %2d) ", ptr[i].x, ptr[i].y);
    printf("\n");
}

int main(void)
{
    FILE* fp;
    Coordinate* xy;
    int* num;
    int count, coord_count;

    fp = fopen("org.dat", "wb");
    num = (int*) malloc(sizeof(int) * 128);
    srand(time(NULL));
    for (int i = 0; i < 128; i++)
        num[i] = rand() % 128 - 64;
    fwrite(num, sizeof(int), 128, fp);
    free(num);
    fclose(fp);

    fp = fopen("org.dat", "rb");
    fseek(fp, 0, SEEK_END);
    count = ftell(fp) / sizeof(int);
    coord_count = count / 2;
    num = (int*) malloc(sizeof(int) * count);

    xy = (Coordinate*) malloc(sizeof(Coordinate) * coord_count);
```

```
    rewind(fp);
    fread(num, sizeof(int), count, fp);
    fclose(fp);

    int valid = 0;
    fp = fopen("output.txt", "w");
    for (int i = 0; i < count; i += 2)
    {
        if (num[i] > 0 && num[i + 1] > 0)
        {
            xy[valid].x = num[i];
            xy[valid].y = num[i + 1];
            valid++;
        }
    }
    printf("valid points:\n");
    display(xy, valid);
    printf("total: %d\n", valid);

    int k, n, pivot;
    printf("k = ");
    scanf("%d", &k);
    printf("n = ");
    scanf("%d", &n);

    pivot = 0;
    while (pivot < valid)
    {
        point.x = xy[pivot].x;
        point.y = xy[pivot].y;
        pivot++;
        selection_sort(xy, pivot, valid);
        printf("the points with a distance of less than %d to (%2d, %2d) are:\n", n, point.x,
point.y);
        if (distance(&xy[pivot + k], &point) < (double)n)
        {
            while (pivot < valid && distance(&xy[pivot], &point) < (double)n)
            {
                printf("(%2d, %2d) %7.2f\n", xy[pivot].x, xy[pivot].y, distance(&xy[pivot],
&point));
                fprintf(fp, "(%2d, %2d) %7.2f\n", xy[pivot].x, xy[pivot].y, distance(&xy[pivot],
&point));
                pivot++;
            }
        }
    }

    fclose(fp);
}
```

输出：

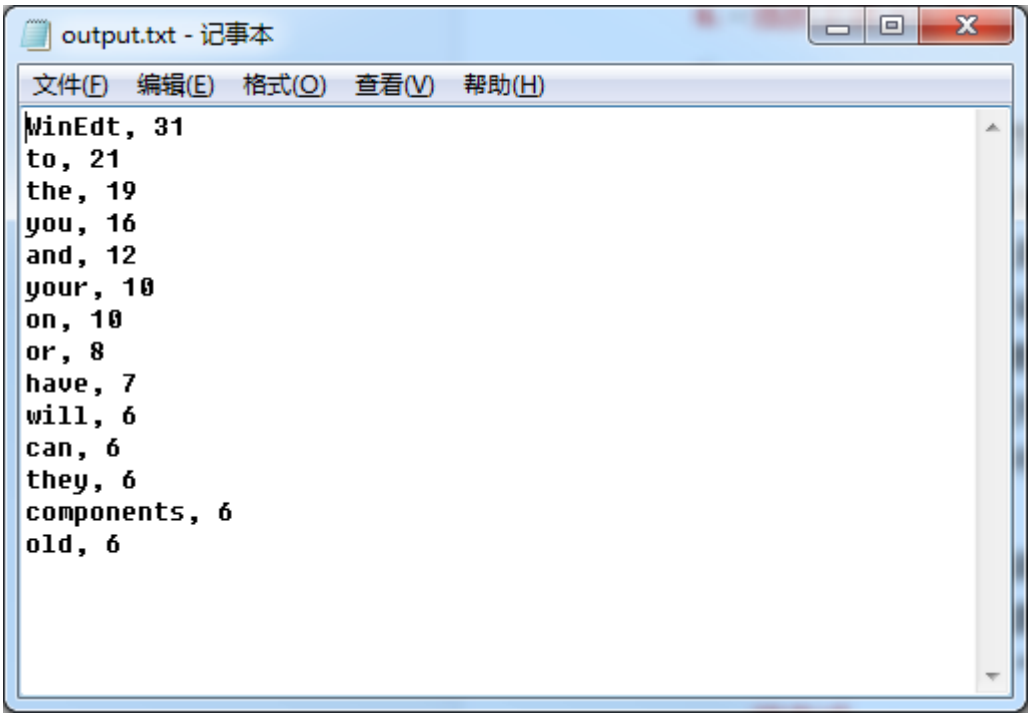
```
valid points:
```

```
(11,  2) (26,  6) (24, 19) ( 8, 50) ( 2, 18) (60, 32) ( 4,  7) (50, 21) (21,  6) (34, 37) (30,
40) ( 5, 16) (31, 59) (60, 34) (18, 32)
total: 15
k = 3
n = 30
the points with a distance of less than 30 to (11,  2) are:
( 4,  7)    8.60
(21,  6)   10.77
( 5, 16)   15.23
(26,  6)   15.52
( 2, 18)   18.36
(24, 19)   21.40
the points with a distance of less than 30 to (18, 32) are:
(30, 40)   14.42
(34, 37)   16.76
( 8, 50)   20.59
(31, 59)   29.97
the points with a distance of less than 30 to (50, 21) are:
the points with a distance of less than 30 to (60, 32) are:
the points with a distance of less than 30 to (60, 34) are:
```

`2016`年复试上机题

要求:

文本文件`input.txt`由若干英文单词和分隔符(空格，回车，换行)构成. 根据如下说明编写程序统计不同单词出现的次数(频度). 将统计结果按出现频度从高到低排序，并将出现频度大于`5`的单词及其频度输出到文件`output.txt`中. 文件格式如图所示



- 多个连续的分隔符被视为一个分隔符.
- 大小写敏感.
- 每个单词的长度不超过`20`个字符.
- 单词的数量未知. 如使用定义静态大数组的方式来统计，将被扣除`5`分.

程序：

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

typedef struct Word
{
    int freq;
    char str[20];
} Word;

int compare(const Word* lhs, const Word* rhs)
{
    if (lhs->freq > rhs->freq) return -1;
    else if (lhs->freq < rhs->freq) return 1;
    else return 0;
}
void swap(Word* e1, Word* e2)
{
    Word tmp = *e1;
    *e1 = *e2;
    *e2 = tmp;
}

void insertion_sort(Word* ptr, int count)
{
    for (int i = 1; i < count; i++)
    {
        int j = i;
        Word ref = ptr[i];
        while (j > 0 && compare(&ptr[j - 1], &ref) > 0)
        {
            ptr[j] = ptr[j - 1];
            j--;
        }
        ptr[j] = ref;
    }
}

int find(Word* words, int count, char* str)
{
    int i;
    for (i = 0; i < count; i++)
        if (strcmp(words[i].str, str) == 0)
            break;
    return i;
}

int main(void)
{
    FILE* fpr = fopen("input.txt", "r");
```

```
    FILE* fpw = fopen("output.txt", "w");
    Word* words;
    char buf[20];
    int count = 0;

    while (fscanf(fpr, "%s", buf) != EOF)
        count++;
    words = (Word*) malloc(sizeof(Word)*count);
    rewind(fpr);
    count = 0;
    while (fscanf(fpr, "%s", buf) != EOF)
    {
        int i = find(words, count, buf);
        if (i == count)
        {
            strcpy(words[count].str, buf);
            words[count].freq = 0;
            count++;
        }
        words[i].freq++;
    }
    insertion_sort(words, count);
    for (int i = 0; i < count && words[i].freq > 5; i++)
    {
        fprintf(fpw, "%s %d\n", words[i].str, words[i].freq);
        printf("%s %d\n", words[i].str, words[i].freq);
    }
    free(words);
    fclose(fpr);
    fclose(fpw);
}
```

输出：

```
that 13
The 11
It 9
to 8
we 8
here 8
a 7
and 6
```

`2017` 年复试上机题

要求：

已知：二进制数据文件 `data.bin` 中存放了若干个整数，请编写程序完成如下功能：

- 编写程序读取所有数据.

- 以每相邻两个整数为一对按顺序构成二维平面上的坐标点. 例如：有数据`12`，`34`，`53`，`25`，`61`，`28`，`78`等，则构成六个坐标点如下：`(12，34)`、`(34，53)`，`(53，25)`，`(25，61)`，`(61，28)`，`(28，78)`；
- 以每个坐标点为圆心，以该点与其后面第一个点的欧氏距离为半径`r`. 计算每个圆包含的坐标点数. 计算最后一个点时以其和第一个点的欧氏距离为半径.

  例如：

  坐标点`(12，34)`的圆半径$r=\sqrt{(12-34)^2+(34-53)^2}$是坐标点`(12，34)`与`(34，53)`的欧式距离.

  坐标点`(28，78)`的圆半径$r=\sqrt{(28-12)^2+(78-34)^2}$是坐标点`(28，78)`与`(12，34)`的欧式距离.
- 计算所有圆的点密度值，然后输出点密度值最大的`5`个坐标点以及相应圆中包含的点数和点密度值. 输出格式要求：

坐标点	包含点数	点密度
(x坐标，y坐标)	(占5列，右对齐)	(占7列，右对齐，保留2位小数)

  上述文字部分不需要显示.

其中：圆的点密度为圆包含的点数除以圆面积，如果点在圆上，则也算圆包含该点，在计算点密度时，圆心也算一个点. 计算圆面积时$\pi=3.14$. 例如：坐标点`(2，1)`，则该坐标点也属该坐标点的圆内的一个点.

程序：

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define PI 3.14

typedef struct
{
    int x;
    int y;
} Coordinate;

typedef struct
{
    double r;
    Coordinate point;
    int points_num;
    double density;
} Circle;

static int compare(const Circle* a, const Circle* b)
{
    if (a->density > b->density) return -1;
    else if (a->density < b->density) return 1;

    else return 0;
```

```
}

static void swap(Circle* a, Circle* b)
{
    Circle tmp = *a;
    *a = *b;
    *b = tmp;
}

void selection_sort(Circle* ptr, int count)
{
    for (int i = 0; i < count; i++)
        for (int j = i + 1; j < count; j++)
            if (compare(&ptr[i], &ptr[j]) > 0)
                swap(&ptr[i], &ptr[j]);
}

int read_file(const char* filename, const char* mode, int*& data)
{
    FILE* fp = fopen(filename, mode);
    int count = 0;

    if (!fp)
    {
        printf("failes to open file\n");
        exit(-1);
    }
    fseek(fp, 0, SEEK_END);
    count = ftell(fp) / sizeof(int);
    data = (int*) malloc(sizeof(int) * count);
    rewind(fp);

    fread(data, sizeof(int), count, fp);
    fclose(fp);
    return count;
}

double distance(Coordinate* a, Coordinate* b)
{
    int val = (a->x - b->x) * (a->x - b->x) +
              (a->y - b->y) * (a->y - b->y);
    return sqrt((double) val);
}

double area(Circle* c)
{
    return (c->r * c->r) * PI;
}

void display(Circle* circles, int count)
{
    for (int i = 0; i < count; i++)

        printf("(%d, %d) %5d %7.2f\n", circles[i].point.x,
```

```
                                        circles[i].point.y,
                                        circles[i].points_num,
                                        circles[i].density);
}

int create_circles(int* data, int count, Circle*& circles)
{
    int circles_num = count - 1;

    circles = (Circle*) malloc(sizeof(Circle) * circles_num);
    for (int i = 0; i < circles_num; i++)
    {
        circles[i].point.x = data[i];
        circles[i].point.y = data[i + 1];
        circles[i].points_num = 0;
    }
    for (int i = 0; i < circles_num; i++)
        circles[i].r = distance(&circles[i].point, &circles[(i+1)%circles_num].point);
    for (int i = 0; i < circles_num; i++)
        for (int j = 0; j < circles_num; j++)
            if (distance(&circles[i].point, &circles[j].point) <= circles[i].r)
                circles[i].points_num++;
    for (int i = 0; i < circles_num; i++)
        circles[i].density = circles[i].points_num / area(&circles[i]);
    return circles_num;
}

int main(void)
{
    int* data = NULL;
    int count = 0;
    int circles_num = 0;
    Circle* circles = NULL;

    count = read_file("data.bin", "rb", data);
    printf("data:\n");
    for (int i = 0; i < count; i++)
        printf("%d ", data[i]);
    printf("\n");
    circles_num = create_circles(data, count, circles);

    selection_sort(circles, circles_num);
    printf("result:\n");
    display(circles, 5);
}
```

输出：

```
data:
2 88 59 83 87 65 38 72 70 76 50 62 4 76 68 70 50 60 13 74 66 60 8 28 97 94 99 52 6 90 69 60 54
83 76 89 64 73 48 69 83 28 84 67 14 50 99 86 35 36 5 82 67 36 92 99 44 27 53 76 24 45 27 19 14
65 86 69 47 80 96 96 10 68 60 91 87 25 15 50 8 18 3 15 85 88 14 8 2 64 63 62 70 58 62 93 51 66
62 73 75 6
result:
(64, 63)     2    0.32
(72, 70)     6    0.05
(63, 62)     9    0.04
(66, 62)    16    0.04
(69, 60)    13    0.04
```

`2016`年保研上机题

要求：

- 请从服务器将两个数据文件`input.txt`和`words.txt`下载到本地电脑的`D`盘根文件夹。
- 在`D`盘根文件夹的`words.txt`中存储了不超过`30000`条的英文单词，每个单词占一行。单词的最大长度为`20`，且单词内部没有空格，文件中无重复单词。
- 在D盘根文件夹的`input.txt`中存储了一个「丢失」了空格和标点符号的英文文章。每行不超过`128`个字符，请编写程序把该文章中第一行和最后一行显示在屏幕上。
- 编写程序将`words.txt`中的最后三行显示在屏幕上；
- 编写程序利用`words.txt`中的单词作为词典，采用正向最大匹配切分单词算法对`input.txt`中的文本进行单词切分。切分时单词区分大小写，切分分割标记采用空格，并将切分后的结果写入到`out.txt`中。

  所谓正向最大匹配切分就是从左向右扫描待切分字符串，尽量取长词。

  下面举一个简单例子：现有待切分字符串`ABCDEFGHIJ`，设词典中最大单词长度为`5`。那么按照算法首先取出`ABCDE`判断是否是单词，如果是则切分到一个单词，否则舍弃最后一个字母接着判断，也就是判断`ABCD`是否是单词，依此类推，当只有一个字母时可以直接认定为是单词。在成功切分出一个单词后对待切分字符串余下的部分再次执行上述过程。
- 编写程序实现步骤`2`、`3`描述的要求，并通过如下所示的主函数对进行验证，注意：除了指定添加的代码之外，不得修改`main`函数其余部分。对`main`函数每修改一处，总分扣`3`分，最多扣`10`分。
- 本次考试考核`C`语言程序设计，因此不可以使用`C++`的`STL`的任何功能，如果需要添加下面样例之外的程序头文件，请举手得到监考老师批准。

程序：

```c
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define SIZE 30000

void showFirstAndLastLine(const char filename[])
{
    char buf[128];
    char temp[128];
    FILE* fp = fopen(filename, "r");
```

```
    if (fgets(buf, 128, fp) != NULL)
        printf("%s", buf);
    while ((fgets(temp, 128, fp) != NULL))
        strcpy(buf, temp);
    fclose(fp);
    printf("%s\n", buf);
}

int ReadWords(const char filename[], char words[SIZE][24])
{
    int count = 0;
    char buf[24];
    FILE* fp = fopen(filename, "r");

    while (fscanf(fp, "%s", buf) != EOF)
        strcpy(words[count++], buf);
    fclose(fp);
    return count;
}

void showTheLastThreeLines(int count, char words[SIZE][24])
{
    printf("%s\n", words[count - 3]);
    printf("%s\n", words[count - 2]);
    printf("%s\n", words[count - 1]);
}

int compare(const void* a, const void* b)
{
    return strcmp((char*) a, (char*) b);
}

void maxMatch(char words[SIZE][24], int count, const char input[], const char output[])
{
    char buf[128];
    char substr[24];
    char* result;
    int i;
    int begin;
    int length;
    FILE* fin = fopen(input, "r");
    FILE* fout = fopen(output, "w");

    qsort(words, count, sizeof(words[0]), compare);
    while (fscanf(fin, "%s", buf) != EOF)
    {
        begin = 0;
        length = strlen(buf);
        while (begin != length)
        {
            memcpy(substr, buf + begin, 20);

            for (i = 20; i > 0; i--)
```

```
            {
                substr[i] = '\0';
                if (bsearch(substr, words, count, sizeof(substr), compare))
                    break;
            }
            fprintf(fout, "%s ", substr);
            begin += strlen(substr);
        }
        fprintf(fout, "\n");
    }
    fclose(fin);
    fclose(fout);
}

int main()
{
    char words[SIZE][24];
    clock_t start, finish;
    start = clock();

    showFirstAndLastLine("input_2016b.txt");
    //将input文件的第一行和最后一行显示在屏幕上

    int count = ReadWords("words_2016b.txt", words);
    //读取英文字典，并返回单词数量

    showTheLastThreeLines(count,  words);
    //此处可以添加处理逻辑以实现题目的第3点要求
    //建议将需要的功能实现为多个函数后在此直接或者间接调用
    maxMatch(words, count, "input_2016b.txt", "output_2016b.txt");

    finish = clock();
    printf("Total time:%lf\n", (double)(finish - start) / CLOCKS_PER_SEC);
}
```

输出：

```
whenIwasyoung
ithinkth
competitivemechanism
internationalization
ling
Total time:0.031250
```

`2017` 年保研上机题

要求：

文本文件 `input.txt` 里面放了一堆整数，仅仅以空格分隔，其中第一个整数为 `k`，第二个整数代表维度，剩余的整数作为点的坐标.

要求：

- 读取整数`k`和维度，并读取剩余的整数组成指定维度的点.
- 从所有点中找到距离最近的两个点并输出它们的坐标和距离.
- 分别输出距离这两个点最近的`k`个点的坐标 .

程序：

```
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <float.h>

typedef struct
{
    int* vector;
    int dimension;
} Point;

Point point;

double distance(Point* a, Point* b)
{
    double dist = 0.0;

    for (int i = 0; i < a->dimension; i++)
        dist += pow(fabs(a->vector[i] - b->vector[i]) * 1.0, 2);
    return sqrt(dist);
}

void swap(Point* a, Point* b)
{
    Point temp = *a;

    *a = *b;
    *b = temp;
}

int compare(Point* a, Point* b)
{
    double dista = distance(a, &point);
    double distb = distance(b, &point);

    if (dista < distb)
        return -1;
    if (dista > distb)
        return 1;
    else
        return 0;
}

void selection_sort(Point* points, int begin, int end)

{
```

```
    for (int i = begin; i < end; i++)
        for (int j = i + 1; j < end; j++)
            if (compare(&points[i], &points[j]) > 0)
                swap(&points[i], &points[j]);
}

void sort_by_distance(Point* points, int begin, int end, int pivot)
{
    point = points[pivot];
    selection_sort(points, begin, end);
}

void display(Point* p)
{
    printf("(");
    for (int j = 0; j < p->dimension - 1; j++)
        printf("%2d, ", p->vector[j]);
    printf("%2d)", p->vector[p->dimension - 1]);
}

void display(Point* points, int count)
{
    for (int i = 0; i < count; i++)
        display(&points[i]);
    printf("\n");
}

void display_nearest(Point* points, int count, int pivot, int k)
{
    sort_by_distance(points, 0, count, pivot);
    printf("the nearest %d points to ", k);
    display(&points[0]);
    printf(":\n");
    for (int i = 1; i <= k; i++)
        display(&points[i]);
    printf("\n");
}

int main()
{
    FILE* fp = fopen("input_2017b.txt", "r");
    Point* points;
    int k, dimension, val, count, point_count;

    count = 0;
    while (fscanf(fp, "%d", &val) != EOF)
        count++;
    rewind(fp);

    fscanf(fp, "%d", &k);
    fscanf(fp, "%d", &dimension);
    printf("k = %d\ndimension = %d\n", k, dimension);
```

```
    point_count = (count - 2) / dimension;
    points = (Point*) malloc(sizeof(Point) * point_count);
    for (int i = 0; i < point_count; i++)
    {
        points[i].dimension = dimension;
        points[i].vector = (int*) malloc(sizeof(int) * dimension);
        for (int j = 0; j < dimension; j++)
            fscanf(fp, "%d", &points[i].vector[j]);
    }
    display(points, point_count);

    int lhs, rhs, pivot;
    double min = distance(&points[0], &points[1]);

    for (pivot = 0; pivot < point_count - 1; pivot++)
    {
        sort_by_distance(points, pivot + 1, point_count, pivot);
        if (distance(&points[pivot], &points[pivot + 1]) < min)
        {
            lhs = pivot;
            rhs = pivot + 1;
            min = distance(&points[pivot], &points[pivot + 1]);
        }
    }
    printf("the shortest distance: %f\n", min);
    printf("the nearest two points:\n");
    display(&points[lhs]);
    display(&points[rhs]);
    printf("\n");

    display_nearest(points, point_count, lhs, k);
    display_nearest(points, point_count, 1, k);
    fclose(fp);
}
```

输出：

```
k = 4
dimension = 3
( 1,  2,  3)( 4,  5,  6)( 7,  8,  9)(10, 11, 12)( 1,  1,  1)( 1,  0,  0)( 1,  0,  1)( 1,  2,  1)(
4,  1,  2)( 0,  0,  0)
the shortest distance: 1.000000
the nearest two points:
( 1,  2,  1)( 1,  1,  1)
the nearest 4 points to ( 1,  2,  1):
( 1,  1,  1)( 1,  2,  3)( 1,  0,  1)( 1,  0,  0)
the nearest 4 points to ( 1,  1,  1):
( 1,  2,  1)( 1,  0,  1)( 1,  0,  0)( 0,  0,  0)
```