Experiment7-董皓彧

环境：

```
gcc.exe (x86_64-win32-seh-rev0, Built by MinGW-W64 project) 8.1.0
Visual Stdio Code 1.83.1
```

作业仓库地址：
https://github.com/FHYQ-Dong/Tsinghua-Program-Design-Assignments/tree/main/Experiment7

必做题

Experiment7-1

题目：

```
求积分：
```

输入格式：

```
无
```

输出格式：

```
一行，一个 4 位小数，为积分结果
```

代码：

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

double f(double x);
double S(double a, double b, double m);

int main() {
    int a = -1, b = 1;
    double m = 10000;
    printf("%.4lf",S(a, b, m));
    return 0;
}

double f(double x) {
    return exp(x * x * (-1));
}

double S(double a, double b, double m) {
    double h = (b - a) / m;
    double sum = 0;
    for (int i = 1; i < m; i++) sum += f(a + i * h);
```

```
    return (f(a) + f(b) + 2 * sum) * h / 2;
}
```

输入1：

```

```

输出1：

```
1.4936
```

Experiment7-2

题目：

```
有 100 头水牛和 100 捆干草，站着的小水牛每头吃了 5 捆草，躺着的小水牛每头吃三捆草，3 头老水牛共吃 1 捆干草。编程求解站着的水牛、躺着的水牛以及老水牛各有多少头（已知每种牛都存在）
```

输入格式：

```
无
```

输出格式：

```
若干行，每行 3 个数，为一组可能解，依次为站着的水牛数、躺着的水牛数、老水牛数
```

代码：

```
#include<stdio.h>

int main() {
    for(int i=1; i<100; i++) {
        for(int j=1; j+i<100; j++) {
            if (((100-i-j) % 3 == 0) && (5*i + 3*j + (100-i-j)/3 == 100)) {
                printf("%d %d %d\n", i, j, 100-i-j);
            }
        }
    }
    return 0;
}
```

输入1：

```

```

输出1：

```
4 18 78
8 11 81
12 4 84
```

选做题

Optional-Experiment7-1

题目：

编程计算 1000 的阶乘（1000!）有多少位。

输入格式：

无

输出格式：

一行，一个整数，为 1000! 的位数

代码：

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

int main() {
    double cnt;
    for(int i=1; i<=1000; ++i) cnt += log10((double)(i));
    printf("%.0lf", cnt);
    return 0;
}
```

输入1：

输出1：

```
2568
```

Optional-Experiment7-2

题目：

已知食品店罐头堆成 n 层，每层排成一个长方形，底层长和宽两边分别为 a 和 b 个罐头，以后每上一层，长和宽两边的罐头各少一个，编程计算当输入 a=1800，b=760，n=10 时罐头的总数。

输入格式：

无

输出格式：

一行，一个整数，总罐头数

代码：

```
#include<stdio.h>

int main() {
    int a = 1800, b = 760, n = 10;
    int sum = 0;
    for(int i=0; i<n; i++) {
        sum += (a - i) * (b - i);
    }
    printf("%d", sum);
}
```

输入1：

```

```

输出1：

```
13565085
```

Optional-Experiment7-3

题目：

有一个六位数 abcdef，由六个不同的数字构成。它的两倍、三倍均为六位数，且这两个六位数都是 a，b，c，d，e，f 这六个数字的某个排列。编程求所有满足条件的六位数。

输入格式：

无

输出格式：

若干行，每行为一个符合要求的六位数

代码：

```
#include<stdio.h>
typedef int bool;
#define true 1
#define false 0

int a[3628800+1];
int cnt, times2, times3;
bool visit[10];

void bsort(int* a, int n) {
    int i, j, temp;
```

```
    for(i = 0; i < n; i++) {
        for(j = 0; j < n - i - 1; j++) {
            if(a[j] > a[j+1]) {
                temp = a[j+1];
                a[j+1] = a[j];
                a[j] = temp;
            }
        }
    }
    return;
}

bool check(int x, int y) {
    int digitx[6], digity[6];
    for(int i = 0; i < 6; i++) {
        digitx[i] = x % 10;
        digity[i] = y % 10;
        x /= 10; y /= 10;
    }
    bsort(digitx, 6); bsort(digity, 6);
    for(int i = 0; i < 6; i++) if(digitx[i] != digity[i]) return false;
    return true;
}

void all_permutation(int ans, int depth) {
    if (depth == 6 && ans >= 100000) {
        a[++cnt] = ans;
        return;
    }
    for(int i=0; i<10; i++) {
        if(!visit[i]) {
            visit[i] = true;
            all_permutation(ans*10+i, depth+1);
            visit[i] = false;
        }
    }
    return;
}

int main() {
    all_permutation(0, 0);
    for(int i=1; i<=cnt; ++i) {
        times2 = a[i] * 2; times3 = a[i] * 3;
        if (times2 > 9999999 || times3 > 9999999) continue;
        if (check(a[i], times2) && check(a[i], times3)) {
            printf("%d\n", a[i]);
        }
    }
    return 0;
}
```

输入1：

```
```

输出1：

```
142857
285714
```

Optional-Experiment7-4

题目：

> 地图上有 A，B，C，D，E 五个国家，如图2所示。现在要对这五个国家着色，要求是相邻的国家必须着不同的颜色。请问至少需要几种颜色才能满足题目要求？并给出一种着色方案，表明每个国家的颜色，颜色用 1，2，3，… 表示。

输入格式：

> 无

输出格式：

> 共两行：
> 第一行：一个整数，最小颜色数量；
> 第二行：5 个整数，为一种可能的着色方案，分别为 A、B、C、D、E 的颜色

代码：

```
#include<stdio.h>
typedef int bool;
#define true 1
#define false 0
typedef struct Border {
    int c1, c2;
} Border;

int color[6] = {0, 1, 2, 3, 4, 5};
int country_color[6]; // A B C D E
Border border[8] = {
    {0, 0}, {1, 3}, {1, 4}, {1, 5},
    {2, 3}, {2, 4}, {3, 4}, {4, 5}
};
bool has_anser;

bool check() {
    for(int i=1; i<=7; ++i) if (country_color[border[i].c1] ==
country_color[border[i].c2]) return false;
    return true;
}

void fill_color(int n, int depth) { // n: 颜色数量
    if (has_anser) return;
    if (depth == 5) {
        if (check()) has_anser = true;
        return;
    }
```

```
    for(int i=1; i<=n; ++i) {
        if (has_anser) return;
        country_color[depth+1] = color[i];
        fill_color(n, depth+1);
    }
    return;
}

int main() {
    for(int i=1; i<=5; ++i) {
        has_anser = false;
        fill_color(i, 0);
        if (has_anser) {
            printf("%d\n", i);
            for(int j=1; j<=5; ++j) printf("%d ", country_color[j]);
            break;
        }
    }
    return 0;
}
```

输入1：

```

```

输出1：

```
3
1 1 2 3 2
```

Optional-Experiment7-5

题目：

```
一个三阶方阵是由 1 至 9 的九个数字构成，要求行、列、及对角线之和均为 15，编程找出所有满足条件的方阵的数字组合。
```

输入格式：

```
无
```

输出格式：

```
共若干行：
第一行：一个整数 n，为满足条件的方阵数量；
接下来每四行为一个满足条件的方阵：其中前三行每行 3 个整数，最后一行为空行，作分隔用
```

代码：

```
#include<stdio.h>
#include<stdlib.h>
#define true 1
```

```
#define false 0
typedef int bool;
typedef struct ans ans;

struct ans {
    int value[3][3];
    ans* next;
};

int matrix[3][3];
int cnt;
ans head;

bool check() {
    for (int i=0; i<3; ++i) {
        int sum = 0;
        for (int j=0; j<3; ++j) sum += matrix[i][j];
        if (sum != 15) return false;
    }
    for (int i=0; i<3; ++i) {
        int sum = 0;
        for (int j=0; j<3; ++j) sum += matrix[j][i];
        if (sum != 15) return false;
    }
    if (matrix[0][0] + matrix[1][1] + matrix[2][2] != 15) return false;
    if (matrix[0][2] + matrix[1][1] + matrix[2][0] != 15) return false;
    return true;
}

void all_permutation(int depth) {
    if (depth == 10) {
        if (check()) {
            ans* tmp = (ans*)malloc(sizeof(ans));
            for(int i=0; i<3; ++i) for(int j=0; j<3; ++j) tmp->value[i][j] =
matrix[i][j];
            tmp->next = head.next;
            head.next = tmp;
            ++cnt;
        }
        return;
    }
    for(int i=0; i<3; ++i) {
        for(int j=0; j<3; ++j) {
            if (matrix[i][j] == 0) {
                matrix[i][j] = depth;
                all_permutation(depth+1);
                matrix[i][j] = 0;
            }
        }
    }
    return;
}

int main() {
    all_permutation(1);
```

```
    ans* tmp = head.next;
    printf("%d\n", cnt);
    while(tmp != NULL) {
        for(int i=0; i<3; ++i) {
            for(int j=0; j<3; ++j) printf("%d ", tmp->value[i][j]);
            printf("\n");
        }
        printf("\n");
        tmp = tmp->next;
    }
    return 0;
}
```

输入1：

```

```

输出1：

```
8
4 9 2
3 5 7
8 1 6

2 9 4
7 5 3
6 1 8

4 3 8
9 5 1
2 7 6

2 7 6
9 5 1
4 3 8

8 3 4
1 5 9
6 7 2

6 7 2
1 5 9
8 3 4

8 1 6
3 5 7
4 9 2

6 1 8
7 5 3
2 9 4
```