第二次课预习练习-函数递归题解

--by lr580

A-复杂函数

将题给函数翻译成代码即可

考察递归入门，基本函数的实现/调用(如绝对值、根号、最值)

一个热知识是 π 通常用 `acos(-1)` 计算，即 $\pi=\arccos(-1)$

参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef double db;
db pi = acos(-1.0), x;
db f(db x)
{
    if (x >= 0.0 && x <= 5.0)
    {
        return sqrt(x) + pi;
    }
    else if (x > 5)
    {
        return min(f(x - 1) + 1.0 / 3 * f(x - 2), 0.5 * f(x - 4));
    }
    else
    {
        return 2 * abs(f(x + 3) * f(x + 4));
    }
}
signed main()
{
    scanf("%lf", &x);
    printf("%lf", f(x));
    return 0;
}
```

B-水桶

这也是一道翻译题，你只需要把自然语言逐句转化为代码即可过题

这题体现了函数的直接和间接递归调用，两个操作会不断套娃，直到区间不存在为止

假设先实现操作 1 ，此时还没有代码实现操作 2 ，却需要调用操作 2 ，所以可以在实现操作 1 之前先声明操作 2 ；这里体现了函数声明的作用

这题没有卡 `long long` ，为降低难度，特地把数据弱化到 `int` 可以过题

参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll s[102], t, v, a, b, k, n;
void f2(ll);
void f1(ll a, ll b)
{
    for (ll i = a; i <= b; ++i)
    {
        ++s[i];
    }
    f2(b);
    if (a + 1 <= b - 1)
    {
        f1(a + 1, b - 1);
    }
}
void f2(ll x)
{
    s[x] += ++k;
    if (x + 2 <= n)
    {
        f1(x + 1, x + 2);
    }
    if (2 * x <= n)
    {
        f2(2 * x);
    }
}
signed main()
{
    scanf("%lld%lld", &n, &t);
    while (t--)
    {
        scanf("%lld", &v);
        if (v == 1)
        {
            scanf("%lld%lld", &a, &b);
            f1(a, b);
        }
        else
        {
            scanf("%lld", &a);
            f2(a);
        }
        for (ll i = 1; i <= n; ++i)
        {
            printf("%lld ", s[i]);
        }
        printf("\n");
    }
    return 0;
}
```

C-最大公因数

> 预告：先修班第二次课会用到最大公因数，如果这道题没做的可以先做一下

正解是使用辗转相除法(欧几里得算法)，该方法你们在数学必修三应该都学过，就不赘述具体步骤了

质因数分解不能过本题，这是因为质因数分解的时间复杂度是 $\mathrm{O}(\sqrt{n})$ ，即假设真的是一个质数，起码要遍历到 $\sqrt{n}$ 次才能结束循环

注意这题卡 `long long` ，所有相关的变量都应该设为 `long long`

> 热知识： 编译器自带 `__gcd` 函数。

代码如下：

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll a, b;
ll gcd(ll a, ll b)
{
    return b ? gcd(b, a % b) : a;
}
signed main()
{
    cin >> a >> b;
    cout << gcd(a, b);
    return 0;
}
```

D-裴蜀定理

> 预告：先修班第二次课会用到裴蜀定理，如果这道题没做的可以先做一下

题目已经告诉了拓展欧几里得算法的具体步骤，所以只需要将其翻译成代码即可。

当 $a' = \gcd(a, b)$ 时，根据欧几里得算法的代码，应该返回 a' ，那么此时便可以设 $x = 1, y = 0$ ；由于每一次新的 x, y 都是根据上一步辗转相除得到的，所以可以在递归函数的递归部分的下方紧接着实现这个操作：

$$x = y', y' = x' - \lfloor \frac{a}{b} \rfloor y'$$

可以把 x, y 设全局变量，也可以设指针参数，也可以设 C++ 传引用，即一种作用类似于指针的语法。

拓展欧几里得算法可以同时计算得到 $\gcd(a, b), x, y$ ，但是本题不需要使用 $\gcd(a, b)$ ，所以不返回这个值也可以。

同样地，这题卡 `long long` 。

下面给出两种细节略有差别的实现(均使用传引用)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll a, b, x, y;
void exgcd(ll a, ll b, ll &x, ll &y)
{
    if (!b)
    {
```

```
        x = 1, y = 0;
        return;
    }
    exgcd(b, a % b, x, y);
    ll t = x;
    x = y;
    y = t - (a / b) * y;
}
signed main()
{
    scanf("%lld%lld", &a, &b);
    exgcd(a, b, x, y);
    printf("%lld %lld", x, y);
    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll a, b, x, y;
void exgcd(ll a, ll b, ll &x, ll &y)
{
    if (b == 0)
    {
        x = 1, y = 0;
        return;
    }
    exgcd(b, a % b, y, x);
    y -= a / b * x;
}
signed main()
{
    scanf("%lld%lld", &a, &b);
    exgcd(a, b, x, y);
    printf("%lld %lld", x, y);
    return 0;
}
```

E-汉诺塔

这题解题思路较为巧妙。

显然 $n=1$ 直接移动即可；所以从 $n=2$ 开始考虑，不难得出，此时只需要 `A->B,A->C,B->C` 即可。

由于每次只能移动顶部圆盘，所以采用一种整体法的思维，将某个柱子的圆盘分解为两部分，即底部的一个圆盘和该柱上面剩下的全部圆盘，类比 $n=2$，假设我们知道剩下的全部圆盘怎么移动，那么只需要：先把上面剩下的全部圆盘移到过渡柱子，然后底部圆盘移动到终点柱子，然后再把上面剩下的全部圆盘移到终点柱子。而 “剩下的全部圆盘” 怎么移动，就是一个 $n'=n-1$ 时的子问题了，因为它恰好就是大小 1 到 $n-1$ 的盘。这意味着，当已知 $n-1$ 的移动方案时，便可以根据上面的做法推知 n 的移动方案。而且这种做法不会产生违背“小圆盘不能放在大圆盘"的条件。

> 如果反过来设状态，即：拆分为顶部一个和剩下全部，那么无法将原问题分解为子问题。

以 $n=3$ 为例，根据上面思路第一步就是先把头两个盘移动到 `B` ，起点是 `A` ，显然不借助过渡是不行的，所以可以拿 `C` 来过渡；即第一步是起点为 `A` ，终点为 `B` ，过渡为 `C` 的子问题。接着第二步，直接把 `A` 底部圆盘移动到 `C` 。接着第三步，把 `B` 的两个圆盘移动到 `C` ，借助 `A` 过渡。第一步和第三步都可以直接化用 $n=2$ 时的答案，只需要重新把 `A,B,C` 改成对应起点、过渡、终点柱子即可。

因此，可以定义操作 $f(n,a,b,c)$ ，代表 n 个圆盘要从起点柱子 a ，借助过渡柱子 b 移动到终点柱子 c 。那么有：

$$f(n,a,b,c)=f(n-1,a,c,b)+\ A->C\ +f(n-1,b,a,c)$$

当 $n>1$ 时，都需要如此执行； $n=1$ 时，直接移动即可，即：

$$f(1,a,b,c)=A->C$$

因此，代码如下：

```c
#include <stdio.h>
void hanio(char a, char b, char c, int n)
{
    if (n > 1)
        hanio(a, c, b, n - 1);
    printf("%c->%c\n", a, c);
    if (n > 1)
        hanio(b, a, c, n - 1);
}
int main()
{
    int n;
    scanf("%d", &n);
    hanio('A', 'B', 'C', n);
    return 0;
}
```

F-打印十字图

注意到起始长度是 5 ，之后每次 n 增加 1 ，两边各增大 2 ，所以总长度为 $5+4n$ ，因此数组每个维度范围不应该小于 $5+4\times 30$ ，且一开始可以直接先铺满这个区域全部是 `.` ，之后每次画用 `$` 替代 `.` 即可。

使用递归来做的话， 可以设当前区域左上角下标为 (x,y) ，横坐标从上往下，纵坐标从左往右。观察可知，除了最中心的小十字之外，其他每一层的大十字都很有规律 (其实中间的十字也可以用相同的规律生成)。假设先不考虑中心小十字：

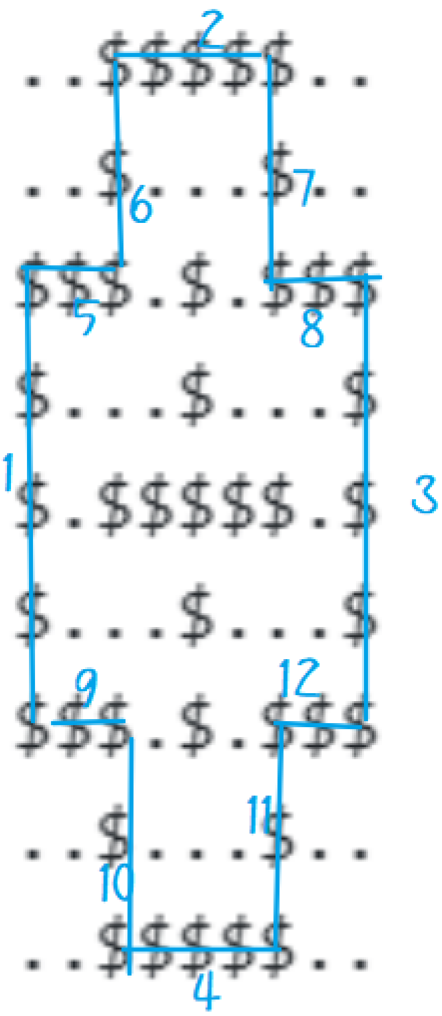
$n\geq 1$ 时，当前区域跨度是 $5+4n$ ，可以以 (x,y) 为左上角，画长为 $1+4n$ 的四条直线和四个内折的角（转化为八条直线）。设 $l=4n$

则这么一个大十字可具体为：

1. 端点在 $(x+2,y),(x+2+l,y)$ 的直线
2. 端点在 $(x,y+2),(x,y+2+l)$ 的直线
3. 端点在 $(x+2,y+4+l),(x+2+l,y+4+l)$ 的直线
4. 端点在 $(x+4+l,y+2),(x+4+l,y+2+l)$ 的直线
5. 端点在 $(x+2,y),(x+2,y+2)$ 的直线
6. 端点在 $(x,y+2),(x+2,y+2)$ 的直线
7. 端点在 $(x,y+2+l),(x+2,y+2+l)$ 的直线

8. 端点在 $(x+2, y+2+l), (x+2, y+4+l)$ 的直线
9. 端点在 $(x+2+l, y), (x+2+l, y+2)$ 的直线
10. 端点在 $(x+2+l, y+2), (x+4+l, y+2)$ 的直线
11. 端点在 $(x+2+l, y+2+l), (x+4+l, y+2+l)$ 的直线
12. 端点在 $(x+2+l, y+2+l), (x+2+l, y+4+l)$ 的直线

如图所示：



可以设一个绘制直线的函数，然后每次画第 i 个大十字复用 12 次，以减轻工作量。

那么，设 $f(n)$ 是绘制第 n 个大十字，有：

$$f(n) = f(n-1) + \text{十二条直线}$$

特别地，根据上面的十二条直线，我们发现若 $n=0, l=1$，恰能对应最中心的小十字。由此，可得 f 定义域为 $n \geq 0$。

绘制直线函数可以抽象为画矩形函数(长或宽为 1)，从而避免横竖的分类讨论。

绘制直线和递归函数的具体实现参见代码：

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
char a[131][131];
void line(ll ax, ll ay, ll bx, ll by)
{
    for (ll i = ax; i <= bx; ++i)
    {
```

```
        for (ll j = ay; j <= by; ++j)
        {
            a[i][j] = '$';
        }
    }
}
void draw(ll x, ll y, ll l, ll n)
{
    if (n < 0)
    {
        return;
    }
    line(x + 2, y, x + 2 + l, y);
    line(x, y + 2, x, y + 2 + l);
    line(x + 2, y + 4 + l, x + 2 + l, y + 4 + l);
    line(x + 4 + l, y + 2, x + 4 + l, y + 2 + l);
    line(x + 2, y, x + 2, y + 2);
    line(x, y + 2, x + 2, y + 2);
    line(x, y + 2 + l, x + 2, y + 2 + l);
    line(x + 2, y + 2 + l, x + 2, y + 4 + l);
    line(x + 2 + l, y, x + 2 + l, y + 2);
    line(x + 2 + l, y + 2, x + 4 + l, y + 2);
    line(x + 2 + l, y + 2 + l, x + 4 + l, y + 2 + l);
    line(x + 2 + l, y + 2 + l, x + 2 + l, y + 4 + l);
    draw(x + 2, y + 2, l - 4, n - 1);
}
ll n, l;
signed main()
{
    cin >> n;
    l = 5 + 4 * n;
    for (ll i = 0; i < l; ++i)
    {
        for (ll j = 0; j < l; ++j)
        {
            a[i][j] = '.';
        }
    }
    draw(0, 0, 4 * n, n);
    for (ll i = 0; i < l; ++i)
    {
        puts(a[i]);
    }
    return 0;
}
```

G-南蛮图腾

这题考察使用递归绘制分形。观察样例：

$n = 1$ 的一个最基本的图案是：

```
 /\
/__\
```

$n=2$ 时，把三角形每个顶点用 $n=1$ 的图案代替即得。

$n=3$ 时，把三角形的每个顶点用 $n=2$ 的图案代替即得。

同理...... n 的绘制需要三个 $n-1$ 的绘制。

设横坐标从上往下，纵坐标从左往右。图案大小为 n 时，容易高(横坐标跨度)是 2^n ，发现底(纵坐标跨度)是 2^{n+1} 。

所以以 (x,y) 为左上角绘制一个大小为 $n(n>1)$ 的图案，等效于分别绘制：

- $(x,y+2^{n-1})$ 为左上角，绘制 $n-1$ 图案
- $(x+2^n,y)$ 为左上角，绘制 $n-1$ 图案
- $(x+2^n,y+2^n)$ 为左上角，绘制 $n-1$ 图案

$n=1$ 时，只有八个下标，分别赋值 `/`, `\`, `_` 和即可。

这样生成时，右边可能会有多余的空格；观察发现，第 i 行右端点为 2^n+i ，所以每次输出到这个范围就停止即可。不要再输出右边多余的空格了。为了实现这个思路，可以先全铺满空格，只画 `/\_` ，然后第 i 行输出前 2^n+i 个字符即可。

> 题意格式这么表述含糊不是我的锅啊，是洛谷原题，要怪就怪洛谷出题人(逃

$n=10$ 时， $2^{n+1}=2048$， 注意数组大小谨防 RE。

技巧：可以用位运算 `1<<(x)` 快速计算 2^x 。

参考代码：

```cpp
#include <bits/stdc++.h>
#define MAXC 4098
char m[MAXC][MAXC];
inline void bas(int i, int j)
{
    m[i][j + 1] = m[i + 1][j] = '/';
    m[i][j + 2] = m[i + 1][j + 3] = '\\';
    m[i + 1][j + 1] = m[i + 1][j + 2] = '_';
}
inline void spr(int i, int j, int n)
{
    if (n == 1)
    {
        bas(i, j);
        return;
    }
    spr(i + (1 << (n - 1)), j, n - 1);
    spr(i, j + (1 << (n - 1)), n - 1);
    spr(i + (1 << (n - 1)), j + (1 << (n)), n - 1);
}
int n, l;
int main()
{
    memset(m, ' ', sizeof m);
    scanf("%d", &n);
    spr(0, 0, n);
    for (int i = 0; i < (1 << n); i++)
    {
        for (int j = 0; j < (1 << n) + i + 1; j++)
        {
            printf("%c", m[i][j]);
```

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

H-正则问题

涉及到括号的都可以使用递归处理，每发现一层括号，就递归进入下一层，单独处理这个括号内的所有内容。直到最细的一层没有括号为止。

所以对于每一层递归可以：

- 读到 `x` ，令当前计次变量自增
- 读到 `|` ，用当前计次变量更新当前层最值，然后置零计次变量重新计次
- 读到 `(` ，进入下一层，当前计次变量赋值为递归结果
- 读到 `)` ，退出当前层，返回当前层最值
- 读到其他字符 (不管是 EOF 还是什么)，说明输入完了，直接返回

可以在每次递归读入输入，也可以每次递归读入全局已读好的字符串。

具体实现参考代码：

```
#include <bits/stdc++.h>
using namespace std;
int finished;
int dfs()
{
    char x;
    int now = 0, maxv = 0;
    while ((x = getchar()) != EOF && !finished)
    {
        if (x == 'x')
            ++now;
        else if (x == '(')
            now += dfs();
        else if (x == ')')
            break;
        else if (x == '|')
            maxv = max(maxv, now), now = 0;
        else
        {
            finished = true;
            break;
        }
    }
    return max(maxv, now);
}
int main()
{
    printf("%d", dfs());
    return 0;
}
```