**模块化的基本原则是什么，如何评价程序的模块化？**

一、基本原则

1. 模块可分解性：如果一种设计方法提供了把问题分解为子问题的系统化机制，那么它就能降低整个问题的复杂度，从而可以实现一种有效的模块化解决方案。

2. 模块可组装性：如果一种设计方法能把现有的设计构建组装成新系统，那么他就能提供一种并非一切都从头开始做的模块化解决方案。

3. 模块可理解性：如果可以把模块作为一种独立单元来理解，那么这样的模块是易于构造、易于修改的。

4. 模块连续性：如果对系统需求的微小修改只导致对个别模块，而不是整个系统的修改，修改引起的副作用将最小。

5. 模块保护性：如果当一个模块内出现异常情况时，它的影响局限在模块内部，错误引起的副作用将最小。

模块的独立性的较好的评价标准是高内聚、低耦合。

1. 程序的模块化评价

1. 提高可维护性：模块化设计将代码分解为相对独立的模块，使得每个模块都有明确的功能和责任。这样的设计使得维护变得更容易，当需要修改代码时，只需关注受影响的模块，而不需要理解整个系统的复杂性。

2. 增加可重用性：模块化将功能划分为独立的模块，可以在不同的项目或场景中重复使用。已有的模块经过测试和验证，可以提高开发效率，减少重复工作，并提高整体代码质量。

3. 促进团队合作：模块化使多个开发人员可以并行开发不同的模块，提高开发效率。每个模块可以由不同的团队成员负责，降低团队合作的复杂性，并允许团队成员专注于自己擅长的领域。

4. 减少错误传播：模块化设计使得单个模块之间的依赖性降低，当出现错误时，只需修复或替换受影响的模块，而无需修改整个系统。这种低耦合的设计有助于降低错误传播的风险。

5. 提高可测试性：模块化使得单元测试和集成测试更加容易。每个模块可以独立地进行测试，确保其功能的正确性。这样可以减少测试所需的时间和资源，提高代码的质量和可靠性。

**如何改变案例程序，使其可以产生任意整数数值范围、任意个二元运算的习题？实现程序，并测试：1.产生50道【0，200】的加减二元运算；2.产生50道【-100，100】的加法或减法的二元运算。**

条件一： 产生任意整数数值范围，通过生成的算式函数里的min 和 max 参数进行设置。

条件二： 产生任意个二元运算的习题，通过改变EQUATION\_NUMBER12 常量进行控制。

【代码如示】

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#define EQUATION\_NUMBER 12 // 这里可以改变生成的算式的个数，即题目中的任意一个

#define COLUMN\_NUMBER 6 // 规定每行的输出格式

typedef struct Equation\_Srtu

{

short int left\_operand, right\_operand; // 这里因为不涉及正负，所以不用unsigned

char operators;

short int value;

} Equation;

// 定义数据结构 其中包括两个左右值 + 符号 + 结果

typedef Equation Exercise[EQUATION\_NUMBER];

void ExerciseGeneratorOfAddition(int min, int max); //包装加法算式 ，加之输出等操作

void ExerciseGeneratorOfSubtracts(int min, int max); //包装减法算式 ，加之输出等操作

short int calculatrAdditionEquation(Equation equation); //计算加法结果

short int calculatrSubstactEquation(Equation equation); //计算减法结果

void generateExerciseOfAdditionEquations(Exercise equationArray, int number, int min, int max); // 生成加法计算 + 包装是否已经存在雷同

void generateExerciseOfSubtractEquations(Exercise equationArray, int number, int min, int max); // 生成减法计算 + 包装是否已经存在雷同

Equation generateAdditionEquation(int min, int max); //具体生成加法算式

Equation generateSubtractEquation(int min, int max); //具体生成减法算式

// 使用min & max 实现题设中的不限数值范围

Equation constructAdditionEquation(short int left, short int right, char op); // 另一种通过建造成的算式的方法

Equation constructSubtractEquation(short int left, short int right, char op);

Equation generateEquation(int min, int max); // 因为随机生成不同的加减计算，所以要有一个包罗加减随机生成加法或减法算式

Equation constructEquation(int min, int max);

int isEqual(Equation eq1, Equation eq2);

int occursIn(Equation anEquation, Exercise equationArray);

void formatAndDisplayExercise(Exercise anExercise, int columns); //格式化打印：每行打印5道含结果的算式

void asString(Equation eq, char\* str);

int main() {

srand(time(NULL)); // 初始化随机数生成器

ExerciseGeneratorOfAddition(0, 200);

ExerciseGeneratorOfSubtracts(0, 200);

ExerciseGeneratorOfAddition(-100, 100);

ExerciseGeneratorOfSubtracts(-100, 100);

return 0;

}

//包装加法算式 ，加之输出等操作

void ExerciseGeneratorOfAddition(int min, int max) {

Exercise anExercise;

generateExerciseOfAdditionEquations(anExercise, EQUATION\_NUMBER, min, max);

//格式化打印：每行打印5道含结果的算式

formatAndDisplayExercise(anExercise, COLUMN\_NUMBER);

}

//包装减法算式 ，加之输出等操作

void ExerciseGeneratorOfSubtracts(int min, int max) {

Exercise anExercise;

generateExerciseOfSubtractEquations(anExercise, EQUATION\_NUMBER, min, max);

formatAndDisplayExercise(anExercise, COLUMN\_NUMBER);

}

//计算加法结果

short int calculatrAdditionEquation(Equation equation) {

return equation.left\_operand + equation.right\_operand;

}

//计算减法结果

short int calculatrSubstactEquation(Equation equation) {

return equation.left\_operand - equation.right\_operand;

}

// 生成加法计算 + 包装是否已经存在雷同

void generateExerciseOfAdditionEquations(Exercise equationArray, int number, int min, int max)

{

Equation equation;

for (int i = 0; i < number; i++)

{

do

{

equation = generateAdditionEquation(min, max);

}

while (occursIn(equation, equationArray));

equationArray[i] = equation;

}

}

//具体生成加法算式

Equation generateAdditionEquation(int min, int max)

{

Equation anEquation;

// 随机生成左操作数和右操作数（在0到100之间）

anEquation.left\_operand = rand() % (max - min + 1) + min;

anEquation.right\_operand = rand() % (max - min + 1) + min;

anEquation.operators = '+';

anEquation.value = calculatrAdditionEquation(anEquation); // 计算结果

return anEquation;

}

// 构造一个加法或减法算式

Equation constructAdditionEquation(short int left, short int right, char op)

{

Equation anEquation;

anEquation.left\_operand = left;

anEquation.right\_operand = right;

anEquation.operators = op;

if (op == '+')

{

anEquation.value = calculatrAdditionEquation(anEquation);

}

else if (op == '-')

{

anEquation.value = calculatrSubstactEquation(anEquation);

}

return anEquation;

}

// 比较两个算式是否相等

int isEqual(Equation eq1, Equation eq2)

{

return (eq1.left\_operand == eq2.left\_operand &&

eq1.right\_operand == eq2.right\_operand &&

eq1.operators == eq2.operators &&

eq1.value == eq2.value

);

}

// 检查一个算式是否已经存在于练习中

int occursIn(Equation anEquation, Exercise equationArray)

{

for (int i = 0; i < EQUATION\_NUMBER; i++)

{

if (isEqual(anEquation, equationArray[i]))

{

return 1;

}

}

return 0;

}

// 格式化打印练习

void formatAndDisplayExercise(Exercise anExercise, int columns) {

for (int i = 0; i < EQUATION\_NUMBER; i++)

{

char str[100];

asString(anExercise[i], str);

printf("%-20s", str);

if ((i + 1) % columns == 0)

{

printf("\n");

}

}

printf("\n");

printf("答案如下：\n");

for(int i=0;i< EQUATION\_NUMBER;i++){

printf("%d\t",anExercise[i].value);

}

printf("\n\n");

}

// 将算式转换为字符串

void asString(Equation eq, char\* str) {

snprintf(str, 100, "%d %c %d = ", eq.left\_operand, eq.operators, eq.right\_operand);

}

// int snprintf(char\* str, size\_t size, const char\* format, ...);

//该函数的作用是将根据指定的 format 字符串格式化的数据写入到字符数组 str 中，

//最多写入 size-1 个字符，然后在末尾添加一个空字符 ‘\0’。

//如果格式化后的字符串长度超过了 size-1，则字符串将被截断，但仍然保证以空字符结尾。

// 生成减法算式练习

void generateExerciseOfSubtractEquations(Exercise equationArray, int number, int min, int max)

{

Equation equation;

for (int i = 0; i < number; i++)

{

do

{

equation = generateSubtractEquation(min, max);

}

while (occursIn(equation, equationArray));

equationArray[i] = equation;

}

}

// 生成减法算式

Equation generateSubtractEquation(int min, int max) {

short int left = rand() % (max - min + 1) + min;

short int right = rand() % (max - min + 1) + min;

char op = '-';

Equation anEquation = constructSubtractEquation(left, right, op);

return anEquation;

}

// 构造一个减法算式

Equation constructSubtractEquation(short int left, short int right, char op) {

Equation anEquation;

anEquation.left\_operand = left;

anEquation.right\_operand = right;

anEquation.operators = op;

anEquation.value = calculatrSubstactEquation(anEquation);

return anEquation;

}

// 生成加法或减法算式

Equation generateEquation(int min, int max)

{

int choice = rand() % 2; // 随机选择是加法还是减法

if (choice == 0)

{

return generateSubtractEquation(min, max);

}

else

{

return generateAdditionEquation(min, max);

}

}

// 构造一个加法或减法算式

Equation constructEquation(int min, int max) {

Equation anEquation;

int choice = rand() % 2; // 随机选择是加法还是减法

if (choice == 0)

{

short int left = rand() % 201; // [0, 200]

short int right = rand() % 201; // [0, 200]

char op = '-';

anEquation = constructSubtractEquation(left, right, op);

}

else

{

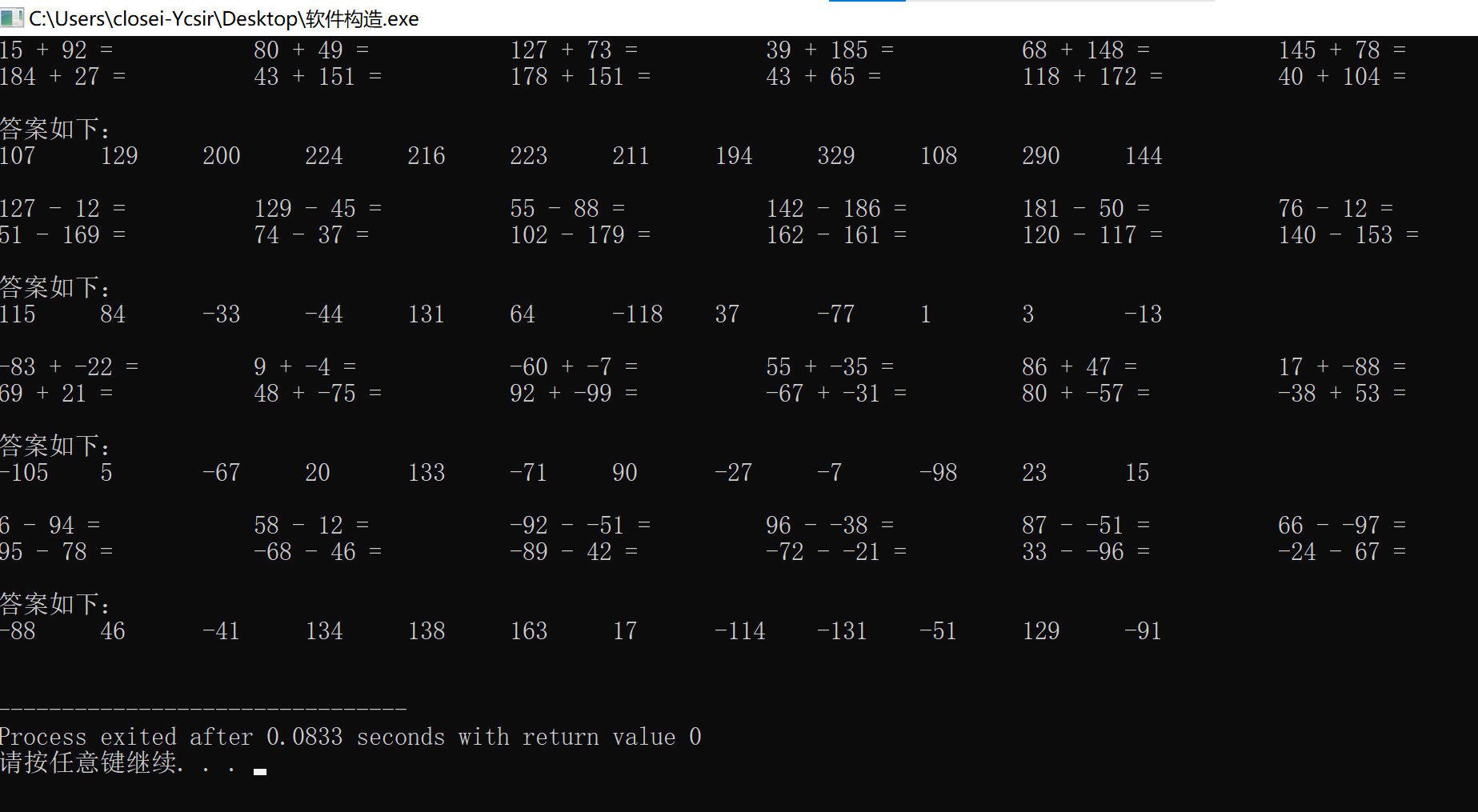
anEquation = generateAdditionEquation(min, max);

}

return anEquation;

}

【结果如示】



函数间的基本逻辑图

