**Лабораторная работа 2. Комбинаторные алгоритмы решения оптимизационных задач**

**Цель работы:** приобрести навыки разработки генераторов подмножеств, перестановок, сочетаний и размещений на С++; научиться применять разработанные генераторы для решения задач о рюкзаке (упрощенную, коммивояжера, об оптимальной загрузке судна и об оптимальной загрузке судна с центровкой.

**ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ:**

**Задание 1.** Разработать генератор подмножеств заданного множества.

// Combi1.h

**#pragma once**

**namespace combi1**

**{**

**struct subset // генератор множества всех подмножеств**

**{**

**short n, // количество элементов исходного множества < 64**

**sn, // количество элементов текущего подмножества**

**\* sset; // массив индексов текущего подмножества**

**unsigned \_\_int64 mask; // битовая маска**

**subset(short n = 1); // конструктор(кол-во эл-ов исх. мн-ва)**

**short getfirst(); // сформ. массив индексов по битовой маске**

**short getnext(); // ++маска и сформировать массив индексов**

**short ntx(short i); // получить i-й элемент массива индексов**

**unsigned \_\_int64 count(); // вычислить общее кол-во подмножеств**

**void reset(); // сбросить генератор, начать сначала**

**};**

**};**

Рис. 1. Шаблон структуры генератора множества всех подмножеств

// Combi1.cpp

**#include "Combi1.h"**

**namespace combi1**

**{**

**subset::subset(short n){**

**this->n = n;**

**this->sset = new short[n];**

**this->reset();**

**};**

**void subset::reset(){**

**this->sn = 0;**

**this->mask = 0;**

**};**

**short subset::getfirst(){**

**\_\_int64 buf = this->mask;**

**this->sn = 0;**

**for (short i = 0; i < n; i++){**

**if (buf & 0x1) this->sset[this->sn++] = i;**

**buf >>= 1;**

**}**

**return this->sn;**

**};**

**short subset::getnext(){**

**int rc = -1;**

**this->sn = 0;**

**if (++this->mask < this->count()) rc = getfirst();**

**return rc;**

**};**

**short subset::ntx(short i){**

**return this->sset[i];**

**};**

**unsigned \_\_int64 subset::count(){**

**return (unsigned \_\_int64)(1 << this->n);**

**};**

**};**

Рис. 2. Реализация методов структуры **subset**

// Main

**#include <iostream>**

**#include "Combi1.h"**

**#include <tchar.h>**

**int \_tmain(int argc, \_TCHAR\* argv[])**

**{**

**setlocale(LC\_ALL, "rus");**

**char AA[][2] = { "A", "B", "C", "D" };**

**std::cout << std::endl << " - Генератор множества всех подмножеств -";**

**std::cout << std::endl << "Исходное множество: ";**

**std::cout << "{ ";**

**for (int i = 0; i < sizeof(AA) / 2; i++)**

**std::cout << AA[i] << ((i < sizeof(AA) / 2 - 1) ? ", " : " ");**

**std::cout << "}";**

**std::cout << std::endl << "Генерация всех подмножеств ";**

**combi1::subset s1(sizeof(AA) / 2); // создание генератора**

**int n = s1.getfirst(); // первое (пустое) подмножество**

**while (n >= 0) // пока есть подмножества**

**{**

**std::cout << std::endl << "{ ";**

**for (int i = 0; i < n; i++)**

**std::cout << AA[s1.ntx(i)] << ((i < n - 1) ? ", " : " ");**

**std::cout << "}";**

**n = s1.getnext(); // cледующее подмножество**

**};**

**std::cout << std::endl << "всего: " << s1.count() << std::endl;**

**system("pause");**

**return 0;**

**}**

Рис. 3. Пример применения генератора множества всех подмножеств

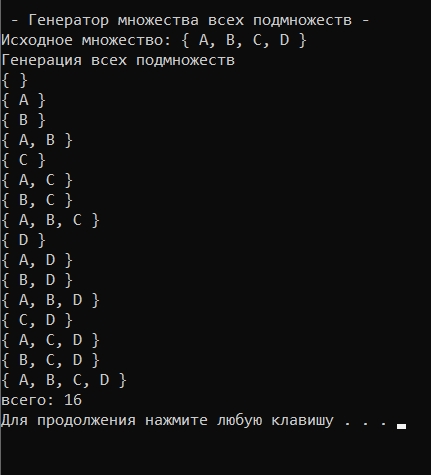


Рис. 4. Вывод результата в консоль

**Задание 2.** Разработать генератор сочетаний.

// Combi2.h

**#pragma once**

**namespace combi2**

**{**

**struct xcombination // генератор сочетаний (эвристика)**

**{**

**short n, // количество элементов исходного множества**

**m, // количество элементов в сочетаниях**

**\* sset; // массив индексов текущего сочетания**

**xcombination(**

**short n = 1, //количество элементов исходного множества**

**short m = 1 // количество элементов в сочетаниях**

**);**

**void reset(); // сбросить генератор, начать сначала**

**short getfirst(); // сформировать первый массив индексов**

**short getnext(); // сформировать следующий массив индексов**

**short ntx(short i); // получить i-й элемент массива индексов**

**unsigned \_\_int64 nc; // номер сочетания 0,..., count()-1**

**unsigned \_\_int64 count() const; // вычислить количество сочетаний**

**};**

**};**

Рис. 5. Шаблон структуры генератора сочетаний

// Combi2.cpp

**#include "Combi2.h"**

**namespace combi2**

**{**

**xcombination::xcombination(short n, short m){**

**this->n = n;**

**this->m = m;**

**this->sset = new short[m + 2];**

**this->reset();**

**}**

**void xcombination::reset() // сбросить генератор, начать сначала**

**{**

**this->nc = 0;**

**for (int i = 0; i < this->m; i++) this->sset[i] = i;**

**this->sset[m] = this->n;**

**this->sset[m + 1] = 0;**

**};**

**short xcombination::getfirst(){**

**return (this->n >= this->m) ? this->m : -1;**

**};**

**short xcombination::getnext() // сформировать следующий массив индексов**

**{**

**short rc = getfirst();**

**if (rc > 0)**

**{**

**short j;**

**for (j = 0; this->sset[j] + 1 == this->sset[j + 1]; ++j) this->sset[j] = j;**

**if (j >= this->m) rc = -1;**

**else {**

**this->sset[j]++;**

**this->nc++;**

**};**

**}**

**return rc;**

**};**

**short xcombination::ntx(short i){return this->sset[i]; };**

**unsigned \_\_int64 fact(unsigned \_\_int64 x) { return(x == 0) ? 1 : (x \* fact(x - 1)); };**

**unsigned \_\_int64 xcombination::count() const**

**{**

**return (this->n >= this->m) ?**

**fact(this->n) / (fact(this->n - this->m) \* fact(this->m)) : 0;**

**};**

**};**

Рис. 6. Реализация функций генератора сочетаний

// Main

**#include <iostream>**

**#include <tchar.h>**

**#include "Combi2.h"**

**int \_tmain(int argc, \_TCHAR\* argv[])**

**{**

**setlocale(LC\_ALL, "rus");**

**char AA[][2] = { "A", "B", "C", "D", "E" };**

**std::cout << std::endl << " --- Генератор сочетаний ---";**

**std::cout << std::endl << "Исходное множество: ";**

**std::cout << "{ ";**

**for (int i = 0; i < sizeof(AA) / 2; i++)**

**std::cout << AA[i] << ((i < sizeof(AA) / 2 - 1) ? ", " : " ");**

**std::cout << "}";**

**std::cout << std::endl << "Генерация сочетаний ";**

**combi2::xcombination xc(sizeof(AA) / 2, 3);**

**std::cout << "из " << xc.n << " по " << xc.m;**

**int n = xc.getfirst();**

**while (n >= 0)**

**{**

**std::cout << std::endl << xc.nc << ": { ";**

**for (int i = 0; i < n; i++)**

**std::cout << AA[xc.ntx(i)] << ((i < n - 1) ? ", " : " ");**

**std::cout << "}";**

**n = xc.getnext();**

**};**

**std::cout << std::endl << "всего: " << xc.count() << std::endl;**

**system("pause");**

**return 0;**

**}**

Рис. 7. Главный файл

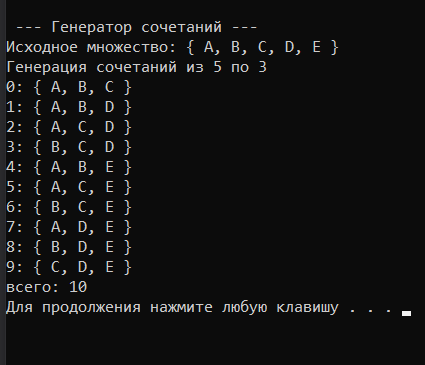


Рис. 8. Вывод результата в консоль

**Задание 3.** Разработать генератор перестановок.

// Combi3.h

**#pragma once**

**namespace combi3**

**{**

**struct permutation // генератор перестановок**

**{**

**const static bool L = true; // левая стрелка**

**const static bool R = false; // правая стрелка**

**short n, // количество элементов исходного множества**

**\* sset; // массив индексов текущей перестановки**

**bool\* dart; // массив стрелок (левых-L и правых-R)**

**permutation(short n = 1); // конструктор (кол-во эл-ов исх. мн-ва)**

**void reset(); // сбросить генератор, начать сначала**

**\_\_int64 getfirst(); // сформировать первый массив индексов**

**\_\_int64 getnext(); // сформировать случайный массив индексов**

**short ntx(short i); // получить i-й элемент масива индексов**

**unsigned \_\_int64 np; // номер перествновки 0,... count()-1**

**unsigned \_\_int64 count() const; // вычислить общее кол. перестановок**

**};**

**};**

Рис. 9. Шаблон структуры генератора перестановок

// Combi3.cpp

**#include "Combi3.h"**

**#include <algorithm>**

**#define NINF ((short)0x8000)**

**namespace combi3**

**{**

**permutation::permutation(short n) {**

**this->n = n;**

**this->sset = new short[n];**

**this->dart = new bool[n];**

**this->reset();**

**};**

**void permutation::reset() { this->getfirst(); };**

**\_\_int64 permutation::getfirst() {**

**this->np = 0;**

**for (int i = 0; i < this->n; i++) { this->sset[i] = i; this->dart[i] = L; };**

**return (this->n > 0) ? this->np : -1;**

**};**

**\_\_int64 permutation::getnext() {**

**\_\_int64 rc = -1;**

**short maxm = NINF, idx = -1;**

**for (int i = 0; i < this->n; i++) {**

**if (i > 0 &&**

**this->dart[i] == L &&**

**this->sset[i] > this->sset[i - 1] &&**

**maxm < this->sset[i]) maxm = this->sset[idx = i];**

**if (i < (this->n - 1) &&**

**this->dart[i] == R &&**

**this->sset[i] > this->sset[i + 1] &&**

**maxm < this->sset[i]) maxm = this->sset[idx = i];**

**};**

**if (idx >= 0) {**

**std::swap(this->sset[idx], this->sset[idx + (this->dart[idx] == L ? -1 : 1)]);**

**std::swap(this->dart[idx], this->dart[idx + (this->dart[idx] == L ? -1 : 1)]);**

**for (int i = 0; i < this->n; i++)**

**if (this->sset[i] > maxm) this->dart[i] = !this->dart[i];**

**rc = ++this->np;**

**}**

**return rc;**

**};**

**short permutation::ntx(short i) { return this->sset[i]; };**

**unsigned \_\_int64 fact(unsigned \_\_int64 x) { return (x == 0) ? 1 : (x \* fact(x - 1)); };**

**unsigned \_\_int64 permutation::count() const { return fact(this->n); };**

**};**

Рис. 10. Реализация функций генератора перестановок

// Main

**#include <iostream>**

**#include "Combi3.h"**

**#include <iomanip>**

**#include <tchar.h>**

**int \_tmain(int argc, \_TCHAR\* argv[])**

**{**

**setlocale(LC\_ALL, "rus");**

**char AA[][2] = { "A", "B", "C", "D" };**

**std::cout << std::endl << " --- Генератор перестановок ---";**

**std::cout << std::endl << "Исходное множество: ";**

**std::cout << "{ ";**

**for (int i = 0; i < sizeof(AA) / 2; i++)**

**std::cout << AA[i] << ((i < sizeof(AA) / 2 - 1) ? ", " : " ");**

**std::cout << "}";**

**std::cout << std::endl << "Генерация перестановок ";**

**combi3::permutation p(sizeof(AA) / 2);**

**\_\_int64 n = p.getfirst();**

**while (n >= 0)**

**{**

**std::cout << std::endl << std::setw(4) << p.np << ": { ";**

**for (int i = 0; i < p.n; i++)**

**std::cout << AA[p.ntx(i)] << ((i < p.n - 1) ? ", " : " ");**

**std::cout << "}";**

**n = p.getnext();**

**};**

**std::cout << std::endl << "всего: " << p.count() << std::endl;**

**system("pause");**

**return 0;**

**}**

Рис. 11. Пример применения генератора перестановок

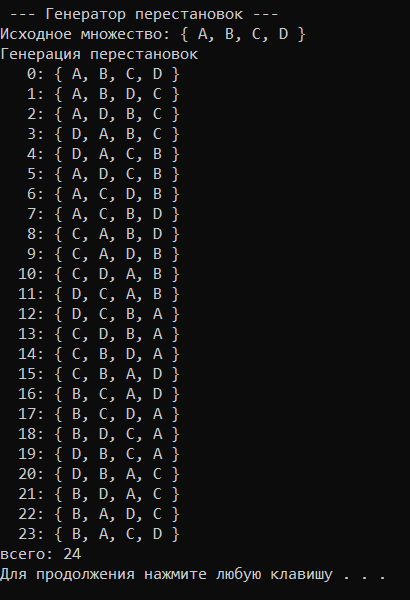


Рис. 12. Вывод результата в консоль

**Задание 4.** Разработать генератор размещений.

// Combi4.h

**#pragma once**

**namespace combi4**

**{**

**struct xcombination // генератор сочетаний (эвристика)**

**{ short n, // кол-во элементов исходного множества**

**m, // количество элементов в сочетаниях**

**\* sset; // массив индексов текущего сочетания**

**xcombination(**

**short n = 1, //количество элементов исходного множества**

**short m = 1 // количество элементов в сочетаниях**

**);**

**void reset(); // сбросить генератор, начать сначала**

**short getfirst(); // сформ. первый массив индексов**

**short getnext(); // сформ. следующий массив индексов**

**short ntx(short i); // получить i-й эл-т массива индексов**

**unsigned \_\_int64 nc; // номер сочетания 0,..., count()-1**

**unsigned \_\_int64 count() const; // вычислить кол-во сочетаний**

**};**

**struct permutation // генератор перестановок**

**{ const static bool L = true; // левая стрелка**

**const static bool R = false; // правая стрелка**

**short n, // количество элементов исходного множества**

**\* sset; // массив индексов текущей перестановки**

**bool\* dart; // массив стрелок (левых-L и правых-R)**

**permutation(short n = 1); // конструктор (кол-во эл-ов исх. мн-ва)**

**void reset(); // сбросить генератор, начать сначала**

**\_\_int64 getfirst(); // сформ. первый массив индексов**

**\_\_int64 getnext(); // сформ. случайный массив индексов**

**short ntx(short i); // получить i-й элемент масива индексов**

**unsigned \_\_int64 np; // номер перествновки 0,... count()-1**

**unsigned \_\_int64 count() const; // вычислить общее кол. перестановок**

**};**

**struct accomodation // генератор размещений**

**{ short n, // количество элементов исходного множества**

**m, // количество элементов в размещении**

**\* sset; // массив индесов текущего размещения**

**xcombination\* cgen; // указатель на генератор сочетаний**

**permutation\* pgen; // указатель на генератор перестановок**

**accomodation(short n = 1, short m = 1); // конструктор**

**void reset(); // сбросить генератор, начать сначала**

**short getfirst(); // сформировать первый массив индексов**

**short getnext(); // сформировать следующий массив индексов**

**short ntx(short i); // получить i-й элемент массива индексов**

**unsigned \_\_int64 na; // номер размещения 0, ..., count()-1**

**unsigned \_\_int64 count() const; // общее количество размещений**

**};**

**}**

Рис. 13. Шаблон структуры генератора размещений

**#include "Combi4.h"**

**#include <algorithm>**

**#define NINF ((short)0x8000)**

**namespace combi4**

**{**

**unsigned \_\_int64 fact(unsigned \_\_int64 x) { return(x == 0) ? 1 : (x \* fact(x - 1)); };**

**xcombination::xcombination(short n, short m) {**

**this->n = n;**

**this->m = m;**

**this->sset = new short[m + 2];**

**this->reset();**

**}**

**void xcombination::reset() // сбросить генератор, начать сначала**

**{**

**this->nc = 0;**

**for (int i = 0; i < this->m; i++) this->sset[i] = i;**

**this->sset[m] = this->n;**

**this->sset[m + 1] = 0;**

**};**

**short xcombination::getfirst() { return (this->n >= this->m) ? this->m : -1; };**

**short xcombination::getnext() // сформировать следующий массив индексов**

**{**

**short rc = getfirst();**

**if (rc > 0) {**

**short j;**

**for (j = 0; this->sset[j] + 1 == this->sset[j + 1]; ++j) this->sset[j] = j;**

**if (j >= this->m) rc = -1;**

**else {**

**this->sset[j]++;**

**this->nc++;**

**};**

**}**

**return rc;**

**};**

**short xcombination::ntx(short i) { return this->sset[i]; };**

**unsigned \_\_int64 xcombination::count() const {**

**return (this->n >= this->m) ?**

**fact(this->n) / (fact(this->n - this->m) \* fact(this->m)) : 0;**

**};**

**permutation::permutation(short n) {**

**this->n = n;**

**this->sset = new short[n];**

**this->dart = new bool[n];**

**this->reset();**

**};**

**void permutation::reset() { this->getfirst(); };**

**\_\_int64 permutation::getfirst() {**

**this->np = 0;**

**for (int i = 0; i < this->n; i++) { this->sset[i] = i; this->dart[i] = L; };**

**return (this->n > 0) ? this->np : -1;**

**};**

**\_\_int64 permutation::getnext() {**

**\_\_int64 rc = -1;**

**short maxm = NINF, idx = -1;**

**for (int i = 0; i < this->n; i++) {**

**if (i > 0 && this->dart[i] == L && this->sset[i] > this->sset[i - 1]**

**&& maxm < this->sset[i]) maxm = this->sset[idx = i];**

**if (i < (this->n - 1) && this->dart[i] == R**

**&& this->sset[i] > this->sset[i + 1]**

**&& maxm < this->sset[i]) maxm = this->sset[idx = i];**

**};**

**if (idx >= 0)**

**{ std::swap(this->sset[idx], this->sset[idx + (this->dart[idx] == L ? -1 : 1)]);**

**std::swap(this->dart[idx], this->dart[idx + (this->dart[idx] == L ? -1 : 1)]);**

**for (int i = 0; i < this->n; i++)**

**if (this->sset[i] > maxm) this->dart[i] = !this->dart[i];**

**rc = ++this->np;**

**}**

**return rc;**

**};**

**short permutation::ntx(short i) { return this->sset[i]; };**

**unsigned \_\_int64 permutation::count() const { return fact(this->n); };**

**accomodation::accomodation(short n, short m) {**

**this->n = n;**

**this->m = m;**

**this->cgen = new xcombination(n, m);**

**this->pgen = new permutation(m);**

**this->sset = new short[m];**

**this->reset();**

**}**

**void accomodation::reset() {**

**this->na = 0;**

**this->cgen->reset();**

**this->pgen->reset();**

**this->cgen->getfirst();**

**};**

**short accomodation::getfirst() {**

**short rc = (this->n >= this->m) ? this->m : -1;**

**if (rc > 0) {**

**for (int i = 0; i <= this->m; i++)**

**this->sset[i] = this->cgen->sset[this->pgen->ntx(i)];**

**};**

**return rc;**

**};**

**short accomodation::getnext() {**

**short rc;**

**this->na++;**

**if ((this->pgen->getnext()) > 0) rc = this->getfirst();**

**else if ((rc = this->cgen->getnext()) > 0)**

**{ this->pgen->reset(); rc = this->getfirst(); };**

**return rc;**

**};**

**short accomodation::ntx(short i) { return this->sset[i]; };**

**unsigned \_\_int64 accomodation::count() const {**

**return (this->n >= this->m) ? fact(this->n) / fact(this->n - this->m) : 0;**

**};**

**}**

Рис. 14. Реализация функций генератора размещений

// main

**#include "stdafx.h"**

**#include <iostream>**

**#include <iomanip>**

**#include <tchar.h>**

**#include "Combi4.h"**

**#define N (sizeof(AA)/2)**

**#define M 3**

**int \_tmain(int argc, \_TCHAR\* argv[])**

**{**

**setlocale(LC\_ALL, "rus");**

**char AA[][2] = { "A", "B", "C", "D" };**

**std::cout << std::endl << " --- Генератор размещений ---";**

**std::cout << std::endl << "Исходное множество: ";**

**std::cout << "{ ";**

**for (int i = 0; i < N; i++)**

**std::cout << AA[i] << ((i < N - 1) ? ", " : " ");**

**std::cout << "}";**

**std::cout << std::endl << "Генерация размещений из " << N << " по " << M;**

**combi4::accomodation s(N, M);**

**int n = s.getfirst();**

**while (n >= 0) {**

**std::cout << std::endl << std::setw(2) << s.na << ": { ";**

**for (int i = 0; i < 3; i++)**

**std::cout << AA[s.ntx(i)] << ((i < n - 1) ? ", " : " ");**

**std::cout << "}";**

**n = s.getnext();**

**};**

**std::cout << std::endl << "всего: " << s.count() << std::endl;**

**system("pause");**

**return 0;**

**}**

Рис. 15. Пример использования генератора перестановок

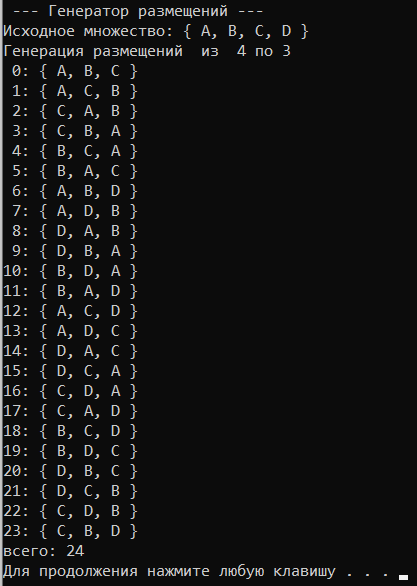


Рис. 15. Пример использования генератора перестановок

**Задание 5.**  Решить в соответствии с вариантом задачу:

* (Вариант 2) упрощенную о рюкзаке (веса предметов и их стоимость сгенерировать случайным образом: вместимость рюкзака 300 кг, веса предметов 10 – 300 кг, стоимость предметов 5 – 55 у.е.; количество предметов – 18 шт.)

//Main.cpp

#include <iostream>

#include "Combi1.h"

#include <tchar.h>

#include "Knapsack.h"

#include "Auxil.h"

#define ITEM\_MAX\_AMOUNT 18 // количество предметов (шт)

#define KNAP\_MAX\_WEIGHT 300 // вместимость рюкзака (кг)

/\*(Вариант 2, 6, 10, 14) упрощенную о рюкзаке (веса предметов и их стоимость сгенерировать случайным образом:

вместимость рюкзака 300 кг, веса предметов 10 – 300 кг, стоимость предметов 5 – 55 у.е.; количество предметов – 18 шт.);\*/

int main()

{

setlocale(LC\_ALL, "rus");

int item\_weight[ITEM\_MAX\_AMOUNT], // вес предмета каждого типа

item\_price[ITEM\_MAX\_AMOUNT]; // стоимость предмета каждого типа

short item\_instance[ITEM\_MAX\_AMOUNT]; // количество предметов каждого типа {0,1}

auxil::start();

for (int i = 0; i < ITEM\_MAX\_AMOUNT; i++) {

item\_weight[i] = (double)auxil::iget(10, 300);

item\_price[i] = (double)auxil::iget(5, 55);

}

int optimal\_weight = knapsack\_s(KNAP\_MAX\_WEIGHT, ITEM\_MAX\_AMOUNT, item\_weight, item\_price, item\_instance);

std::cout << "\t\tУпрощенная задача о рюкзаке";

std::cout << std::endl << "- количество предметов (шт):\t\t" << ITEM\_MAX\_AMOUNT;

std::cout << std::endl << "- вместимость рюкзака (кг):\t\t" << KNAP\_MAX\_WEIGHT;

std::cout << std::endl << "- размеры предметов (кг):\t\t";

for (int i = 0; i < ITEM\_MAX\_AMOUNT; i++) std::cout << item\_weight[i] << " ";

std::cout << std::endl << "- стоимости предмета ($):\t\t";

for (int i = 0; i < ITEM\_MAX\_AMOUNT; i++) std::cout << item\_price[i] << " ";

std::cout << std::endl << "- оптимальная стоимость рюкзака ($):\t" << optimal\_weight;

std::cout << std::endl << "- вес рюкзака (кг):\t\t\t";

int s = 0; for (int i = 0; i < ITEM\_MAX\_AMOUNT; i++) s += item\_instance[i] \* item\_weight[i];

std::cout << s;

std::cout << std::endl << "- выбраны предметы:\t\t\t";

for (int i = 0; i < ITEM\_MAX\_AMOUNT; i++) std::cout << item\_instance[i] << " ";

std::cout << std::endl << std::endl;

system("pause");

return 0;

}

//Knapsack.cpp

#include "Knapsack.h"

#define NINF 0x80000000

int calcv(combi1::subset s, const int v[])

{

int rc = 0;

for (int i = 0; i < s.sn; i++) rc += v[s.ntx(i)];

return rc;

};

int calcc(combi1::subset s, const int v[], const int c[])

{

int rc = 0;

for (int i = 0; i < s.sn; i++) rc += (v[s.ntx(i)] \* c[s.ntx(i)]);

return rc;

};

void setm(combi1::subset s, short m[])

{

for (int i = 0; i < s.n; i++) m[i] = 0;

for (int i = 0; i < s.sn; i++) m[s.ntx(i)] = 1;

};

int knapsack\_s(

int V, // [in]

short n, // [in]

const int v[], // [in]

const int c[], // [in]

short m[] // [out] {0,1}

)

{

combi1::subset s(n);

int maxc = NINF, cc = 0;

short ns = s.getfirst();

while (ns >= 0)

{

if (calcv(s, v) <= V)

if ((cc = calcc(s, v, c)) > maxc)

{

maxc = cc;

setm(s, m);

}

ns = s.getnext();

};

return maxc;

};

//Knapsack.h

#pragma once

#include "Combi1.h"

int knapsack\_s(

int V, // [in] вместимость рюкзака

short n, // [in] количество типов предметов

const int v[], // [in] размер предмета каждого типа

const int c[], // [in] стоимость предмета каждого типа

short m[] // [out] количество предметов каждого типа

);

//Combi1.cpp

#include "Combi1.h"

namespace combi1

{

subset::subset(short n) {

this->n = n;

this->sset = new short[n];

this->reset();

};

void subset::reset() {

this->sn = 0;

this->mask = 0;

};

short subset::getfirst() {

\_\_int64 buf = this->mask;

this->sn = 0;

for (short i = 0; i < n; i++) {

if (buf & 0x1) this->sset[this->sn++] = i;

buf >>= 1;

}

return this->sn;

};

short subset::getnext() {

int rc = -1;

this->sn = 0;

if (++this->mask < this->count()) rc = getfirst();

return rc;

};

short subset::ntx(short i) {

return this->sset[i];

};

unsigned \_\_int64 subset::count() {

return (unsigned \_\_int64)(1 << this->n);

};

};

//Combi1.h

#pragma once

namespace combi1

{

struct subset // генератор множества всех подмножеств

{

short n, // количество элементов исходного множества < 64

sn, // количество элементов текущего подмножества

\* sset; // массив индексов текущего подмножества

unsigned \_\_int64 mask; // битовая маска

subset(short n = 1); // конструктор(кол-во эл-ов исх. мн-ва)

short getfirst(); // сформ. массив индексов по битовой маске

short getnext(); // ++маска и сформировать массив индексов

short ntx(short i); // получить i-й элемент массива индексов

unsigned \_\_int64 count(); // вычислить общее кол-во подмножеств

void reset(); // сбросить генератор, начать сначала

};

};

//Auxil.cpp

#include "Auxil.h"

namespace auxil

{

void start()

{

srand((unsigned)time(NULL));

}

double dget(double rmin, double rmax)

{

return (double)rand() / RAND\_MAX \* (rmax - rmin) + rmin;

}

int iget(int rmin, int rmax)

{

return (int)dget((double)rmin, (double)rmax);

}

}

//Auxil.h

#pragma once

#include <cstdlib>

#include <ctime>

namespace auxil

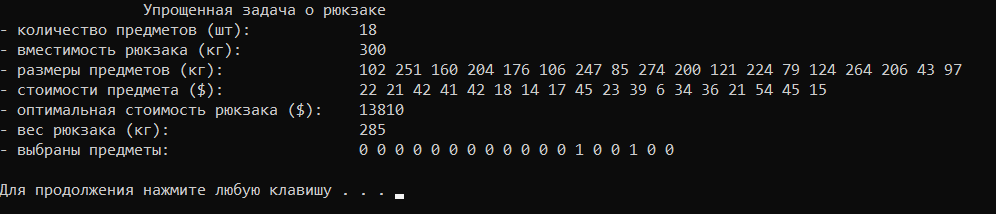
{

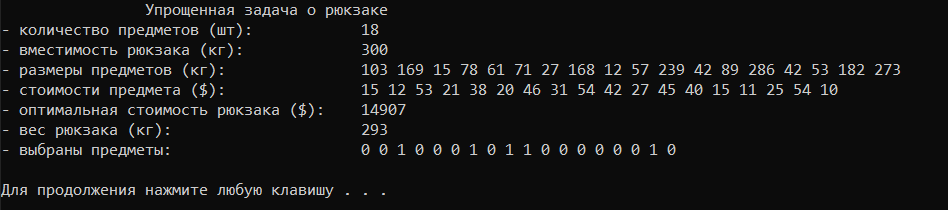
void start();

double dget(double rmin, double rmax);

int iget(int rmin, int rmax);

};





**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи (в соответствии с вариантом) от размерности задачи:

* (Вариант 2, 6, 10, 14) упрощенную о рюкзаке (количество предметов 12 – 20 шт.);

//Main.cpp

#include <iostream>

#include "Combi1.h"

#include "Knapsack.h"

#include <time.h>

#include <tchar.h>

#include <iomanip>

#define ITEM\_MAX\_AMOUNT 20 // количество предметов (шт)

#define KNAP\_MAX\_WEIGHT 300 // вместимость рюкзака (кг)

/\*(Вариант 2, 6, 10, 14) упрощенную о рюкзаке (количество предметов 12 – 20 шт.)\*/

int main() {

setlocale(LC\_ALL, "rus");

int v[] = { 25, 56, 67, 40, 20, 27, 37, 33, 33, 44, 53, 12,

60, 75, 12, 55, 54, 42, 43, 14 },

c[] = { 15, 26, 27, 43, 16, 26, 42, 22, 34, 12, 33, 30,

12, 45, 60, 41, 33, 11, 14, 12 };

short m[ITEM\_MAX\_AMOUNT];

int maxcc = 0;

clock\_t t1, t2;

std::cout << "\tЗадача о рюкзаке\n (исследование времени)\n";

std::cout << std::setfill('-') << std::setw(40) << "\n";

std::cout << "вместимость рюкзака: " << KNAP\_MAX\_WEIGHT << std::endl;

std::cout << std::setfill('-') << std::setw(40) << "\n";

std::cout << std::setfill(' ') << std::setw(15) << std::right << "количество";

std::cout << " | ";

std::cout << std::setfill(' ') << std::setw(20) << std::left << "продолжительность";

std::cout << std::endl << std::setfill(' ') << std::setw(15) << std::right << "предметов";

std::cout << " | ";

std::cout << std::setfill(' ') << std::setw(20) << std::left << "вычисления";

std::cout << std::setfill('-') << std::setw(40) << "\n";

for (int i = 12; i <= ITEM\_MAX\_AMOUNT; i++)

{

t1 = clock();

maxcc = knapsack\_s(KNAP\_MAX\_WEIGHT, i, v, c, m);

t2 = clock();

std::cout << std::endl << std::setfill(' ') << std::setw(15) << std::right << i;

std::cout << " | ";

std::cout << std::setfill(' ') << std::setw(20) << std::left << t2 - t1;

}

std::cout << std::endl << std::endl;

system("pause");

}

//Combi1.cpp

#include "Combi1.h"

namespace combi1

{

subset::subset(short n) {

this->n = n;

this->sset = new short[n];

this->reset();

};

void subset::reset() {

this->sn = 0;

this->mask = 0;

};

short subset::getfirst() {

\_\_int64 buf = this->mask;

this->sn = 0;

for (short i = 0; i < n; i++) {

if (buf & 0x1) this->sset[this->sn++] = i;

buf >>= 1;

}

return this->sn;

};

short subset::getnext() {

int rc = -1;

this->sn = 0;

if (++this->mask < this->count()) rc = getfirst();

return rc;

};

short subset::ntx(short i) {

return this->sset[i];

};

unsigned \_\_int64 subset::count() {

return (unsigned \_\_int64)(1 << this->n);

};

};

//Combi1.h

#pragma once

namespace combi1

{

struct subset // генератор множества всех подмножеств

{

short n, // количество элементов исходного множества < 64

sn, // количество элементов текущего подмножества

\* sset; // массив индексов текущего подмножества

unsigned \_\_int64 mask; // битовая маска

subset(short n = 1); // конструктор(кол-во эл-ов исх. мн-ва)

short getfirst(); // сформ. массив индексов по битовой маске

short getnext(); // ++маска и сформировать массив индексов

short ntx(short i); // получить i-й элемент массива индексов

unsigned \_\_int64 count(); // вычислить общее кол-во подмножеств

void reset(); // сбросить генератор, начать сначала

};

};

//Knapsack.cpp

#include "Knapsack.h"

#define NINF 0x80000000

int calcv(combi1::subset s, const int v[])

{

int rc = 0;

for (int i = 0; i < s.sn; i++) rc += v[s.ntx(i)];

return rc;

};

int calcc(combi1::subset s, const int v[], const int c[])

{

int rc = 0;

for (int i = 0; i < s.sn; i++) rc += (v[s.ntx(i)] \* c[s.ntx(i)]);

return rc;

};

void setm(combi1::subset s, short m[])

{

for (int i = 0; i < s.n; i++) m[i] = 0;

for (int i = 0; i < s.sn; i++) m[s.ntx(i)] = 1;

};

int knapsack\_s(

int V, // [in]

short n, // [in]

const int v[], // [in]

const int c[], // [in]

short m[] // [out] {0,1}

)

{

combi1::subset s(n);

int maxc = NINF, cc = 0;

short ns = s.getfirst();

while (ns >= 0)

{

if (calcv(s, v) <= V)

if ((cc = calcc(s, v, c)) > maxc)

{

maxc = cc;

setm(s, m);

}

ns = s.getnext();

};

return maxc;

};

//Knapsack.h

#pragma once

#include "Combi1.h"

int knapsack\_s(

int V, // [in] вместимость рюкзака

short n, // [in] количество типов предметов

const int v[], // [in] размер предмета каждого типа

const int c[], // [in] стоимость предмета каждого типа

short m[] // [out] количество предметов каждого типа

);

