УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

**Отчёт по лабораторной работе №1**

**по дисциплине «Математическое программирование»**

**на тему «Вспомогательные функции»**

Выполнил:

Студент 2 курса 8 группы

Семёнов Даниил Вячеславович

Преподаватель: асс. Ромыш А.С.

Минск

2025

**ЦЕЛЬ РАБОТЫ:** приобретение навыков составления и отладки программ с использованием пользовательских функций для замера продолжительности процесса вычисления.

Перед выполнением заданий следует подключить предварительно скомпилированный заголовок stdafx.h. Для использования предварительно скомпилированных заголовков их необходимо включить параметром /Y или через страницу свойств проекта **Configuration Properties > C/C++ > Precompiled Headers**. После этого в каждом cpp-файле в самом начале (до любых директив препроцессора или строк кода) нужно добавить #include "stdafx.h".

***Задание 1.*** Разработайте три функции (start, dget и iget), используя следующие спецификации:

**void start(); -** устанавливает начальные числа для генератора псевдослучайных чисел.

**double dget(**

**double rmin,** //[in]минимальное значение

**double rmax** //[in]максимальное значение

**); -** генерирует действительное псевдослучайное число в заданном диапазоне.

**int iget(**

**int rmin,** //[in]минимальное значение

**int rmax** //[in]максимальное значение

**); -** генерирует целое псевдослучайное число в заданном диапазоне.

Кроме того, по заданию разработанные функции должны располагаться в файле **Auxil.cpp**,а в файле **Auxil.h –** прототипы функций. Код из Auxil.cpp можно увидеть в листинге 1.1.

|  |
| --- |
| #include "stdafx.h"  namespace auxil  {  void start() // старт генератора сл. чисел  {  srand((unsigned)time(NULL));  };  double dget(double rmin, double rmax) // получить случайное число  {  return ((double)rand() / (double)RAND\_MAX) \* (rmax - rmin) + rmin;  };  int iget(int rmin, int rmax) // получить случайное число  {  return (int)dget((double)rmin, (double)rmax);  };  } |

Листинг 1.1 – Файл Auxil.cpp

Код из Auxil.h представлен в листинге 1.2.

|  |
| --- |
| #pragma once  #include <cstdlib>  namespace auxil  {  void start(); // старт генератора сл. чисел  double dget(double rmin, double rmax); // получить случайное число  int iget(int rmin, int rmax); // получить случайное число  }; |

Листинг 1.2 – Файл Auxil.h

***Задание 2***

1. Реализовать пример 2.
2. Для проверки работоспособности разработанных функций и приобретения навыков замера продолжительности процесса вычисления реализуйте программу, приведенную в примере 2.

Реализация кода представлена в листинге 2.1.

|  |
| --- |
| #include "stdafx.h"  #define CYCLE 1000000 // количество циклов  int main()  {  double av1 = 0, av2 = 0;  clock\_t t1 = 0, t2 = 0;  setlocale(LC\_ALL, "rus");  auxil::start(); // старт генерации  t1 = clock(); // фиксация времени  for (int i = 0; i < CYCLE; i++)  {  av1 += (double)auxil::iget(-100, 100); // сумма случайных чисел  av2 += auxil::dget(-100, 100); // сумма случайных чисел  }  t2 = clock(); // фиксация времени  std::cout << std::endl << "количество циклов: " << CYCLE;  std::cout << std::endl << "среднее значение (int): " << av1 / CYCLE;  std::cout << std::endl << "среднее значение (double): " << av2 / CYCLE;  std::cout << std::endl << "продолжительность (у.е): " << (t2 - t1);  std::cout << std::endl << " (сек): "  << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC);  std::cout << std::endl;  system("pause");  return 0;  } |

Листинг 2.1 – Файл main.cpp

Результат выполнения программы представлен на рисунке 1.

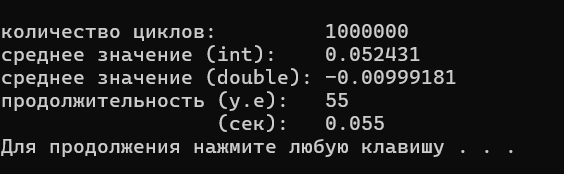


Рисунок 1 – Результат работы программы

**Задание 3**

Проведите необходимые эксперименты и постройте график зависимости (Excel) продолжительности процесса вычисления от количества циклов в примере 2. Проанализируйте характер зависимости. Проведите исследование любого другого рекурсивного алгоритма, например, вычисления факториала или генератора чисел Фибоначчи (прим. – например вычислите каким будет 100-е, 200-е, 300-е и т.д число), и включите в отчет график. Кроме этого продолжительность вычисления следует измерять в условных единицах процессорного времени (функция **clock**).

Перед выполнением следует найти зависимость продолжительности вычислений от количества циклов. Для этого необходимо изменять количество циклов и запоминать время выполнения при этих значениях. Диапазон циклов выбираем от 100000 до 1000000. На рисунке 2 можно увидеть результат вычислений.

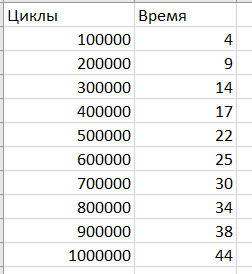


Рисунок 2 – Результаты тестирования №1

На рисунке 3 представлен график зависимости.



Рисунок 3 – График зависимости времени от цикла

Исходя из результатов, мы видим, что зависимость линейная. Теперь проведём исследование с числами Фибоначчи. Перед этим следует реализовать программу.

|  |
| --- |
| #pragma once  int fibonachi(int n); |

Листинг 3.1 – Файл Fibonachi.h

Реализация кода Fibonachi.cpp представлена в листинге 3.2.

|  |
| --- |
| #include "stdafx.h"  using namespace std;  int fibonachi(int n)  {  if (n == 0)  return 0;  if (n == 1)  return 1;  return fibonachi(n - 1) + fibonachi(n - 2);  } |

Листинг 3.1 – Файл Fibonachi.cpp

Обновлённый код main.cpp представлен в листинге 3.3.

|  |
| --- |
| #include "stdafx.h"  #define CYCLE 1000000 // количество циклов  using namespace std;  int main()  {  setlocale(LC\_ALL, "rus");  cout << "\t\tЧисла Фибоначчи" << endl;  int N;  clock\_t t3 = 0, t4 = 0;  cout << "Введите количество чисел N: ";  cin >> N;  t3 = clock();  int result = fibonachi(N);  cout << "Результат функции чисел Фибоначчи: " << result;  t4 = clock();  std::cout << std::endl << "продолжительность (у.е): " << (t4 - t3);  std::cout << std::endl << " (сек): "  << ((double)(t4 - t3)) / ((double)CLOCKS\_PER\_SEC);  std::cout << std::endl;  system("pause");  return 0;  } |

Листинг 3.1 – Файл main.cpp

Результат выполнения программы представлен на рисунке 4.

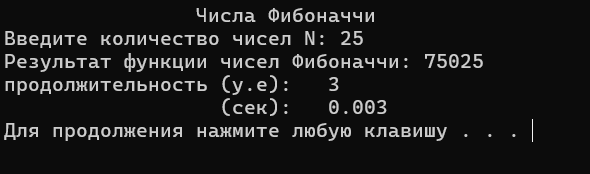


Рисунок 4 – Результат работы программы

Далее определим зависимость продолжительности вычислений от входного параметра N для алгоритма, реализующего числа Фибоначчи.

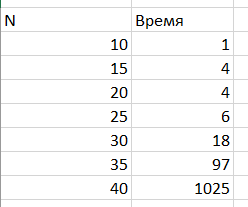


Рисунок 5 – Результаты тестирования №2

Далее по найденным значениям построим график и определим зависимость.

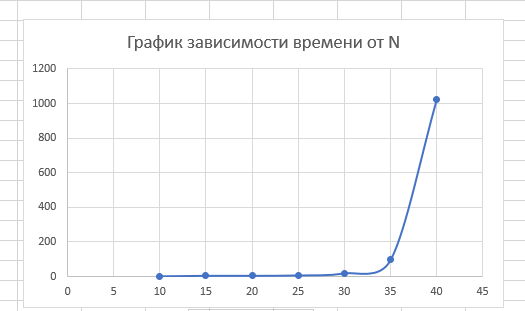


Рисунок 6 – График зависимости времени от N

Исходя из графика можно сделать вывод, что зависимость – экспоненциальная.

Подводя итоги, мы можем утверждать, что зависимость времени выполнения от количества циклов в первом случае линейна, т.к. количество операций прямо пропорционально числу циклов. Во втором эксперименте зависимость выполнения алгоритма Фибоначчи – экспоненциальная, т.к. мы видим резко возрастающий график.

**Лабораторная работа №2**

**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 1.** Разобрать и разработать генератор подмножеств заданного множества.

Генератор подмножеств представляет собой алгоритм или инструмент, предназначенный для формирования всех возможных комбинаций элементов, входящих в состав заданного множества. Подмножество – это совокупность элементов, которые являются частью исходного множества, но могут не включать все его элементы.

Реализация файла Combi.h представлена в листинге 1.1.

|  |
| --- |
| #pragma once  namespace combi  {  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(количество элементов исходного множества)  short getfirst(); // сформормировать 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 количество подмножеств  void reset(); // сбросить генератор, начать сначала  };  }; |

Листинг 1.1 – Файл Combi.h

Реализация файла Combi.cpp представлена в листинге 1.2.

|  |
| --- |
| #include "Combi.h"  #include <algorithm>  namespace combi  {  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);  };  }; |

Листинг 1.2 – Файл Combi.cpp

Реализация файла main.cpp представлена в листинге 1.

|  |
| --- |
| #include <iostream>  #include "Combi.h"  int main() // с помощью инкременирования битовой маски  {  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 ";  combi::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;  } |

Листинг 1.3 – Файл Main.cpp

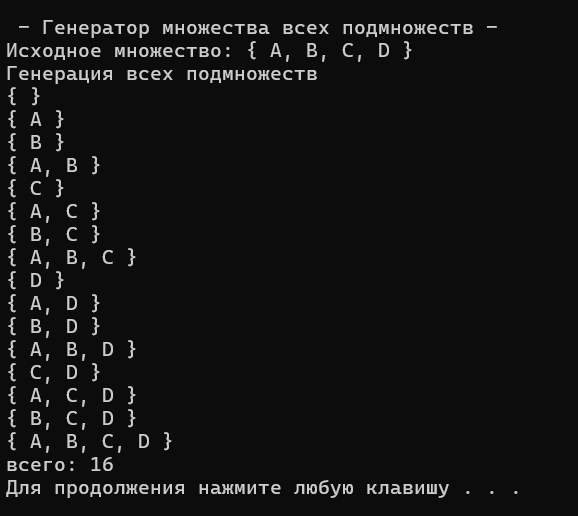


Рисунок 1 – Результат работы программы

**Задание 2.** Разобрать и разработать генератор сочетаний.

Генератор сочетаний — это инструмент или алгоритм, предназначенный для создания всех возможных комбинаций элементов из заданного множества, где каждая комбинация имеет определенный размер. В отличие от перестановок, сочетания не учитывают порядок элементов.

|  |
| --- |
| #pragma once  namespace combi  {  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  };  }; |

Листинг 2.1 – Файл Combi.h

|  |
| --- |
| #include "Combi.h"  #include <algorithm>  namespace combi  {  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;  };  } |

Листинг 2.2 – Файл Combi.cpp

|  |
| --- |
| #include <iostream>  #include "Combi.h"  int main() // тоже битовая маска, но берем только подмножества с N двоичными единицами  {  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 ";  combi::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;  } |

Листинг 2.3 – Файл Main.cpp

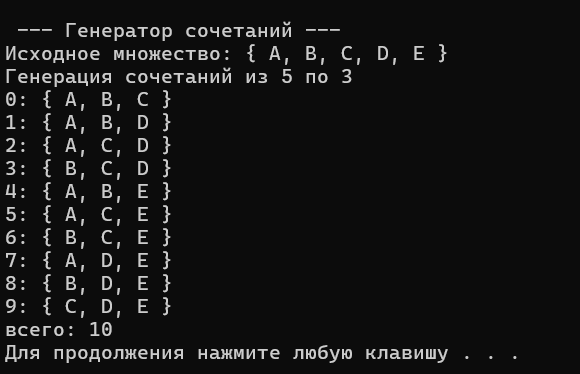


Рисунок 2 – Результат работы программы

**Задание 3.** Разобрать и разработать генератор перестановок.

Генератор перестановок — это инструмент или алгоритм, который формирует все возможные упорядоченные комбинации (перестановки) элементов из заданного множества. Каждая перестановка представляет собой определённый порядок расположения элементов, где последовательность имеет значение.

|  |
| --- |
| #pragma once  namespace combi  {  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 (количество элементов исходного множества)  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  };  }; |

Листинг 3.1 – Файл Combi.h

|  |
| --- |
| #include "Combi.h"  #include <algorithm>  #define NINF ((short)0x8000)  namespace combi  {  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); };  } |

Листинг 3.2 – Файл Combi.cpp

|  |
| --- |
| #include <iostream>  #include "Combi.h"  #include <iomanip>  int main() // алгоритм Джонсона – Троттера  {  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 ";  combi::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;  } |

Листинг 3.3 – Файл Main.cpp

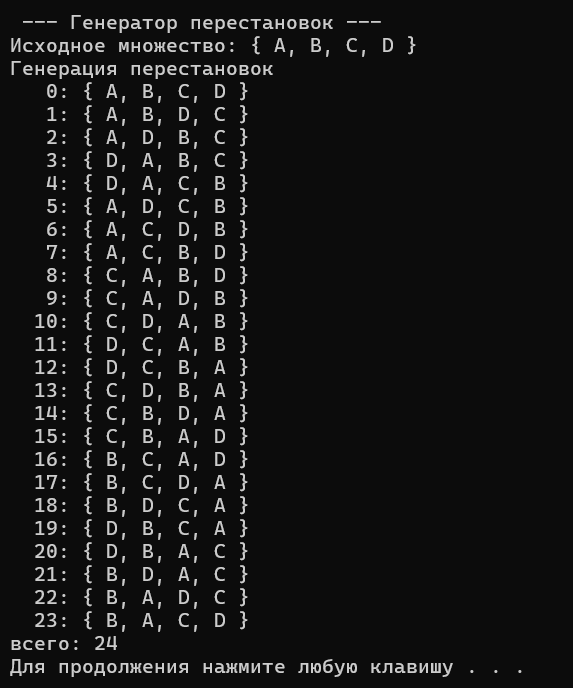


Рисунок 3 – Результат работы программы

**Задание 4.** Разобрать и разработать генератор размещений.

Генератор размещений — это инструмент или алгоритм, предназначенный для создания всех возможных упорядоченных комбинаций (размещений) элементов из заданного множества, где каждая комбинация имеет определённый размер. Размещение учитывает порядок элементов, а также может включать повторяющиеся элементы, если это разрешено. В отличие от сочетаний, где порядок не важен, в размещениях последовательность элементов играет ключевую роль.

|  |
| --- |
| #pragma once  namespace combi  {  struct xcombination  {  short n,  m,  \* sset;  xcombination(  short n = 1,  short m = 1  );  void reset();  short getfirst();  short getnext();  short ntx(short i);  unsigned \_\_int64 nc;  unsigned \_\_int64 count() const;  };  struct permutation  {  const static bool L = true;  const static bool R = false;  short n,  \* sset;  bool\* dart;  permutation(short n = 1);  void reset();  \_\_int64 getfirst();  \_\_int64 getnext();  short ntx(short i);  unsigned \_\_int64 np;  unsigned \_\_int64 count() const;  };  struct accomodation  {  short n,  m,  \* sset;  xcombination\* cgen;  permutation\* pgen;  accomodation(short n = 1, short m = 1);  void reset();  short getfirst();  short getnext();  short ntx(short i);  unsigned \_\_int64 na;  unsigned \_\_int64 count() const;  };  } |

Листинг 4.1 – Файл Combi.h

|  |
| --- |
| #include "Combi.h"  #include <algorithm>  #define NINF ((short)0x8000)  namespace combi  {  // ================================== ACCOMODATION ==================================  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];  };  // факториал из структуры accomodation  unsigned \_\_int64 fact(unsigned \_\_int64 x) { return (x == 0) ? 1 : (x \* fact(x - 1)); };  unsigned \_\_int64 accomodation::count() const  {  return (this->n >= this->m) ?  fact(this->n) / fact(this->n - this->m) : 0;  };  // ================================== PERMUTATION ==================================  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); };  // ================================== XCOMBINATION ==================================  xcombination::xcombination(short n, short m)  {  this->n = n;  this->m = m;  this->sset = new short[m + 2];  this->reset();  }  void xcombination::reset()  {  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()  {  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;  };  } |

Листинг 4.2 – Файл Combi.cpp

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include "Combi.h"  #define N (sizeof(AA)/2)  #define M 3  int main()  {  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;  combi::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 < M; 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;  } |

Листинг 4.3 – Файл Main.cpp

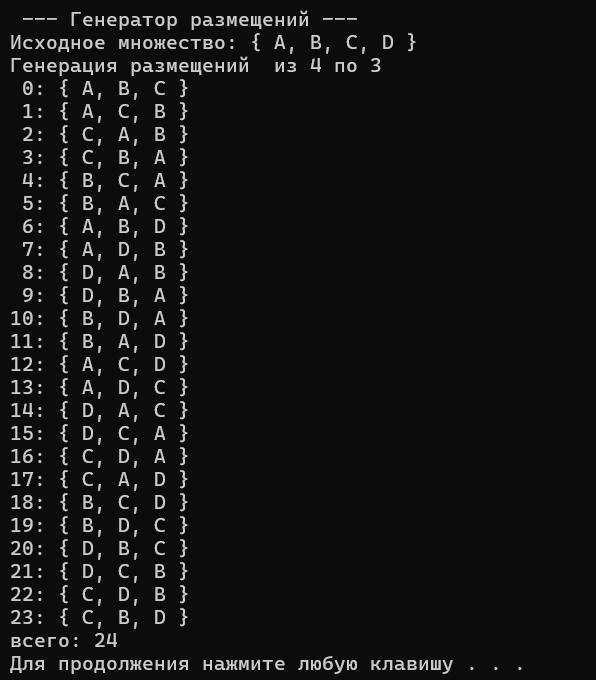


Рисунок 4 – Результат работы программы

**Задание 5.**  Решить в соответствии с вариантом задачу и результат занести в отчет (Вариант распределяется по списку): 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.);

|  |
| --- |
| #pragma once  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  ); |

Листинг 5.1 – Файлknapsack\_s.h

|  |
| --- |
| #include "knapsack\_s.h"  #include "../Subsets/Combi.h"  #define NINF 0x80000000 // самое малое int-число  int calcv(combi::subset s, const int v[]) // объем в рюкзаке  {  int rc = 0;  for (int i = 0; i < s.sn; i++) rc += v[s.ntx(i)];  return rc;  };  int calcc(combi::subset s, const int v[], const int c[]) //стоимость в рюкзаке  {  int rc = 0;  for (int i = 0; i < s.sn; i++) rc += (v[s.ntx(i)] \* c[s.ntx(i)]);  return rc;  };  void setm(combi::subset s, short m[]) //отметить выбранные предметы  {  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] 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 {0,1}  )  {  combi::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;  }; |

Листинг 5.2 – Файл knapsack\_s.cpp

|  |
| --- |
| #include <iostream>  #include "../Subsets/Combi.cpp"  #include "knapsack\_s.h"  #include <ctime>  #include <random>  #define NN 18  int main()  {  setlocale(LC\_ALL, "rus"); // Устанавливаем локаль для корректного отображения русских символов  int\* v = new int[NN]; // Массив для хранения весов предметов  int\* c = new int[NN]; // Массив для хранения стоимостей предметов  int V = 300; // Вместимость рюкзака  srand(time(0)); // Инициализация генератора случайных чисел  // Заполнение массивов случайными значениями  for (int i = 0; i < NN; i++) {  v[i] = rand() % 290 + 10; // Вес предмета от 10 до 300  }  for (int i = 0; i < NN; i++) {  c[i] = rand() % 50 + 5; // Стоимость предмета от 5 до 55  }  short m[NN]; // Массив для хранения выбранных предметов (0 или 1)  clock\_t t1 = 0, t2 = 0;  t1 = clock(); // Начало отсчёта времени  // Вызов функции для решения задачи о рюкзаке  int maxcc = knapsack\_s(  V, // Вместимость рюкзака  NN, // Количество предметов  v, // Массив весов предметов  c, // Массив стоимостей предметов  m // Массив для хранения выбранных предметов  );  t2 = clock(); // Конец отсчёта времени  // Вывод результатов  std::cout << std::endl << "-------- Результат работы программы --------- ";  std::cout << std::endl << "- Количество предметов: " << NN;  std::cout << std::endl << "- Вместимость рюкзака: " << V;  std::cout << std::endl << "- Веса предметов: ";  for (int i = 0; i < NN; i++) std::cout << v[i] << " ";  std::cout << std::endl << "- Стоимости предметов: ";  for (int i = 0; i < NN; i++) std::cout << c[i] << " ";  std::cout << std::endl << "- Максимальная стоимость рюкзака: " << maxcc;  std::cout << std::endl << "- Вес выбранных предметов: ";  int s = 0;  for (int i = 0; i < NN; i++) s += m[i] \* v[i];  std::cout << s;  std::cout << std::endl << "- Выбранные предметы: " << std::endl;  for (int i = 0; i < NN; i++) {  if (m[i] > 0) {  std::cout << "( Стоимость: " << c[i] << ")";  std::cout << "( Вес: " << v[i] << ")";  std::cout << "( id: " << i << ")" << std::endl;  }  }  std::cout << std::endl << std::endl;  std::cout << std::endl << "Затраченное время (мс): " << (t2 - t1);  std::cout << std::endl << " (сек): " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC);  std::cout << std::endl;  system("pause");  return 0;  } |

Листинг 5.3 – Файл main.cpp

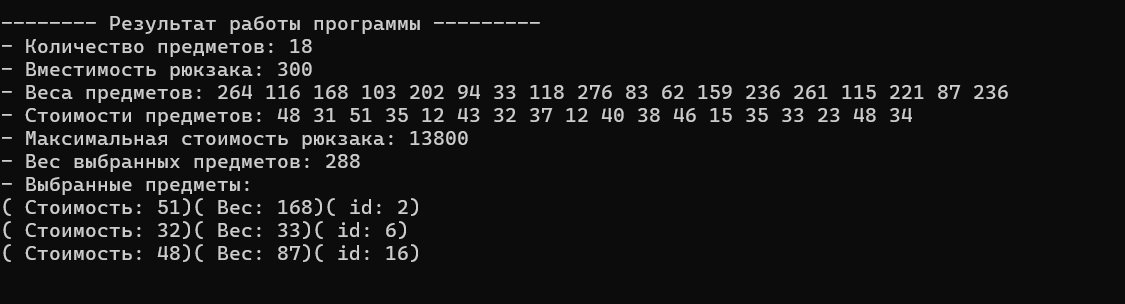


Рисунок 5 – Результат работы программы

**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи (в соответствии с вариантом) от размерности задачи и результат в виде графика с небольшим пояснением занести в отчет: (2, 6, 10, 14) упрощенную о рюкзаке (количество предметов 12 – 20 шт.);

Зависимость времени вычисления от количества предметов представлена на рисунке 6.

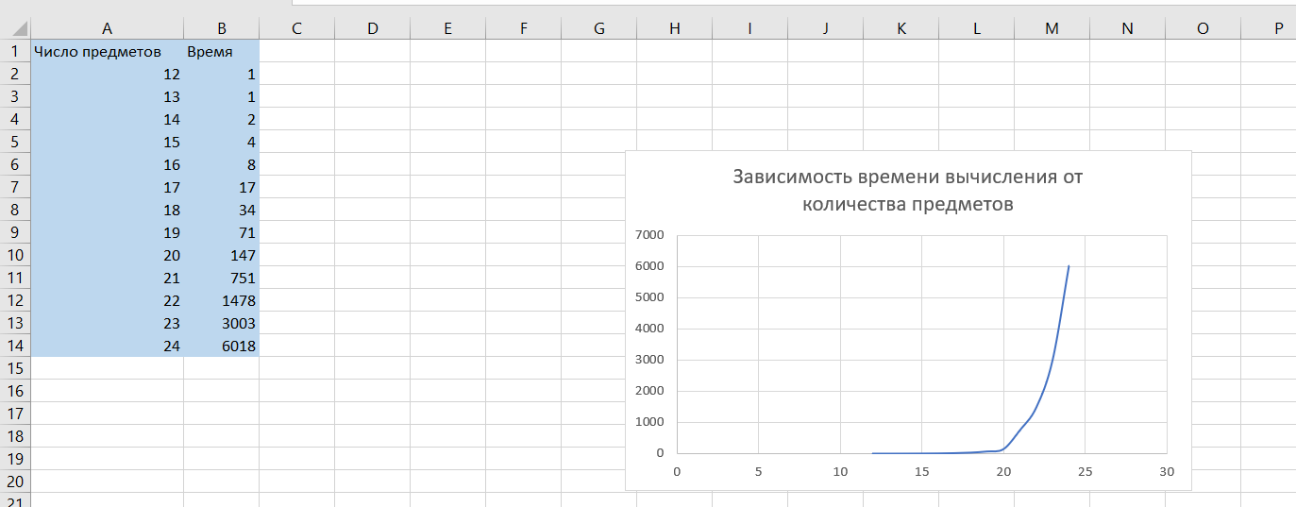


Рисунок 5 – Результат работы программы

Вывод: исходя из полученных данных и графика зависимости можно сделать вывод, что зависимость экспоненциальная.