```
//Программа работает 21-02-2019 г. Визуал Студио С++

//Разложение натурального числа на простые множители

//Cranium  факторизация числа  30 ноября 2015
//Задача: найти разложение натурального числа на простые множители(факторизация числа).

//     Исходный код на языке С++
#include <iostream>
#include <vector>

	using namespace std;

template <typename intT>
vector<intT> fact(intT num) {

	intT j = 2;
	vector<intT> res;
	while (num / intT(2) >= j) {
		if (num % j == 0) {
			res.push_back(j);
			num /= j;
			j = intT(2);
		}
		else {
			++j;
		}
	}
	res.push_back(num);
	return res;
}


//typedef unsigned long integralT;
typedef __int64 integralT;

int main() {

	vector<integralT> v;
	integralT n;

	while (true) {
		cout << "Enter positive number: ";
		cin >> n;
		v.clear();
		v = fact(n);

		for (auto i = v.begin(); i != v.end(); i++) {
			if (i != v.begin())
				cout << ", ";
			cout << *i;
		}
		cout << endl << endl;
	}

	return 0;
}
```

```
//Динамическое программирование
//Динамическое программирование(ДП) – это техника, которая разделяет задачу на маленькие пересекающиеся подзадачи, считает решение для каждой из них и сохраняет его в таблицу.Окончательное решение считывается из таблицы.
//Ключевая особенность динамического программирования – способность определять состояние записей в таблице и отношения или перемещения между записями.
//Затем, определив базовые и рекурсивные случаи, можно заполнить таблицу сверху вниз или снизу вверх.
//В нисходящем ДП таблица будет заполнена рекурсивно, по мере необходимости, начиная сверху и спускаясь к меньшим подзадачам.В восходящем ДП таблица будет заполняться по порядку, начиная с меньших подзадач и с использованием их решений для того чтобы подниматься выше и находить решения для бо́льших задач.В обоих случаях если решение данной подзадачи уже встречалось, оно просто ищется в таблице.И это значительно снижает вычислительные затраты.
//Пример: Биноминальные коэффициенты
//     Мы используем пример биноминальных коэффициентов, чтобы проиллюстрировать использование нисходящего и восходящего ДП.Код ниже основан на рекурсиях для биноминальных коэффициентов с перекрывающимися подзадачами.Обозначим через C(n, k) количество выборок из n по k, тогда имеем :
//Базовый случай : C(n, 0) = C(n, n) = 1
//Рекурсия : C(n, k) = C(n - 1, k - 1) + C(n - 1, k)
//     У нас есть несколько перекрывающихся подзадач.Например, для C(n = 5, k = 2) рекурсивное дерево будет следующим :

//C(5, 2)
// \
//C(4, 1)                              C(4, 2)
// \ / \
//C(3, 0)   C(3, 1)             C(3, 1)             C(3, 2)
// \ / \ / \
//C(2, 0)  C(2, 1)      C(2, 0) C(2, 1)    C(2, 1)  C(2, 2)
// \ / \ / \
//C(1, 0)  C(1, 1)    C(1, 0)  C(1, 1) C(1, 0)  C(1, 1)
//Мы можем реализовать нисходящее и восходящее ДП
//следующим образом :

#include <iostream>
#include <cstring>

using namespace std;

#define V 8

int memo[V][V]; // таблица

int min(int a, int b) { return (a < b) ? a : b; }

void print_table(int memo[V][V])
{
	for (int i = 0; i < V; ++i)
	{
		for (int j = 0; j < V; ++j)
		{
			printf(" %2d", memo[i][j]);
		}
		printf("\n");
	}
}

int binomial_coeffs1(int n, int k)
{
	// Нисходящее ДП
	if (k == 0 || k == n) return 1;
```

```
	if (memo[n][k] != -1) return memo[n][k];
	return memo[n][k] = binomial_coeffs1(n - 1, k - 1) +
		binomial_coeffs1(n - 1, k);
}

int binomial_coeffs2(int n, int k)
{
	// Восходящее ДП
	for (int i = 0; i <= n; ++i)
	{
		for (int j = 0; j <= min(i, k); ++j)
		{
			if (j == 0 || j == i)
			{
				memo[i][j] = 1;
			}
			else
			{
				memo[i][j] = memo[i - 1][j - 1] + memo[i - 1][j];
			}
		}
	}
	return memo[n][k];
}

int main()
{
	int n = 5, k = 2;
	printf("Нисходящее ДП:\n");
	memset(memo, -1, sizeof(memo));
	int nCk1 = binomial_coeffs1(n, k);
	print_table(memo);
	printf("C(n = %d, k = %d): %d\n\n", n, k, nCk1);

	printf("Восходящее ДП:\n");
	memset(memo, -1, sizeof(memo));
	int nCk2 = binomial_coeffs2(n, k);
	print_table(memo);
	printf("C(n = %d, k = %d): %d\n", n, k, nCk2);

	return 0;
}
//
//При C(n = 5, k = 2) код выше выводит следующее :
//Нисходящее ДП :
//-1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
//- 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
//- 1  2 - 1 - 1 - 1 - 1 - 1 - 1
//- 1  3  3 - 1 - 1 - 1 - 1 - 1
//- 1  4  6 - 1 - 1 - 1 - 1 - 1
//- 1 - 1 10 - 1 - 1 - 1 - 1 - 1
//- 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
//- 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
//C(n = 5, k = 2) : 10

//Восходящее ДП :
//1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
//1  1 - 1 - 1 - 1 - 1 - 1 - 1
//1  2  1 - 1 - 1 - 1 - 1 - 1
//1  3  3 - 1 - 1 - 1 - 1 - 1
//1  4  6 - 1 - 1 - 1 - 1 - 1
//1  5 10 - 1 - 1 - 1 - 1 - 1
//- 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
//- 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
```

```
//C(n = 5, k = 2) : 10
//Временная и пространственная сложность будут выражены как O(n * k).
//В случае нисходящего ДП решения подзадач накапливались по мере необходимости, в то время как в восходящем ДП таблица заполнялась начиная с базового случая.
//
//Примечание Для печати был выбран
```

.