Министерство образования Республики Беларусь

Учреждение Образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет Компьютерных Систем и Сетей

Кафедра Информатики

Курсовой проект

на тему:

**«Реализация алгоритма построения кроссворда с использование метода поиска с возвратом»**

Выполнил:

Королёв Д. Ф.

Руководитель:

Козуб В. Н.

2017

**Содержание**

[Введение 3](#_Toc482564247)

[Примеры работы программы 4](#_Toc482564248)

[Техническое описание проекта 8](#_Toc482564249)

[Требования к кроссворду 8](#_Toc482564250)

[Описание метода поиска с возвратом 8](#_Toc482564251)

[Описание работы программы 9](#_Toc482564252)

[Заключение 13](#_Toc482564253)

[Приложениие. 14](#_Toc482564254)

**Введение**

Каждый из нас хотя бы раз встречался с кроссвордами. Они могут занять человека любого возраста на несколько часов. Хотя разгадывание кроссвордов — вещь увлекательная, их составление может оказаться весьма трудоёмким.

Поэтому, меня заинтересовал вопрос построения кроссвордов. И темой своей курсовой работы я выбрал «Реализация алгоритма построения кроссворда с использование метода поиска с возвратом».

**Цель**: разработать и реализовать алгоритм построения кроссворда с помощью метода поиска с возвратом.

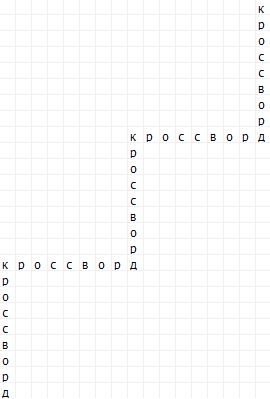
Для достижения цели необходимо решить **следующие задачи:**

1. Выявить оптимальный способ построения кроссворда;
2. Реализовать данный способ средствами языка C++.

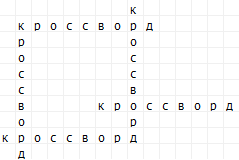
# **Примеры работы программы**

Программа позволяет выбрать предпочитаемый способ расположения слов: максимально отдалённо, случайным образом, максимально плотно.

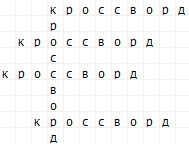
* 5 слов «кроссворд», максимально отдалённо:

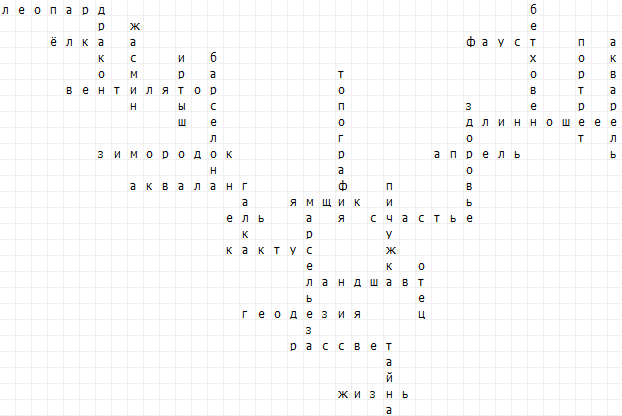


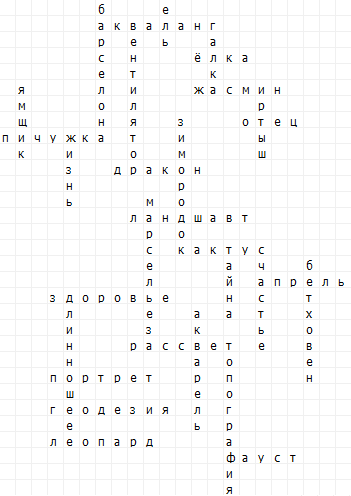
* 5 слов «кроссворд», случайным образом:



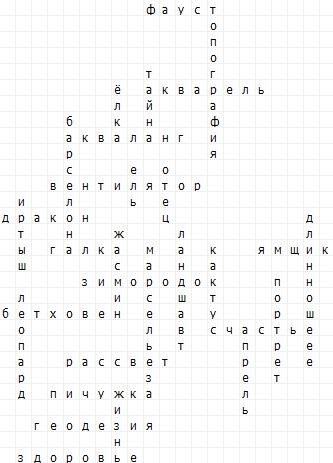
* 5 слов «кроссворд», максимально плотно:



* 30 слов, максимально отдалённо:
* 30 слов, случайным образом

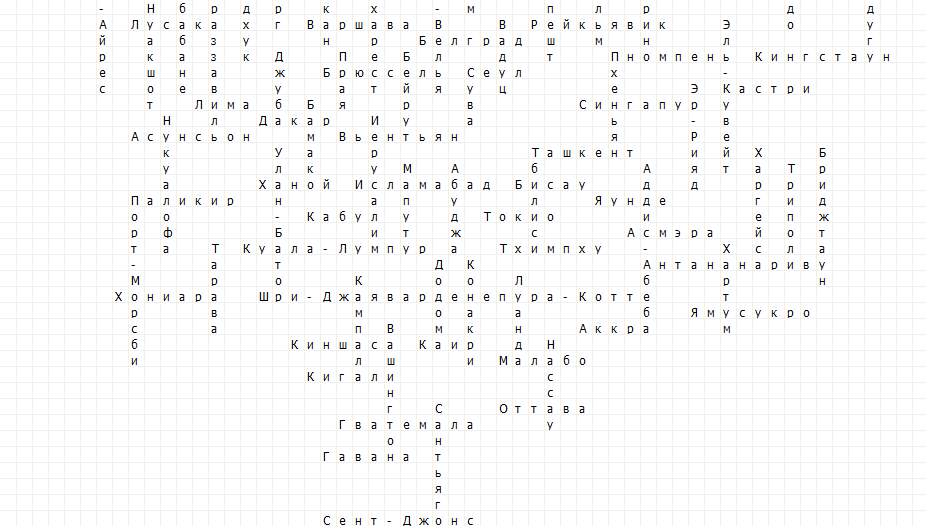


* 30 слов, максимально плотно:



* 198 слов (столицы стран мира), максимально плотно (2 части):





# **Техническое описание проекта**

## **Требования к кроссворду**

1. Слова в кроссворде располагаются в виде цепочки ячеек, в каждую из которых по порядку выписываются буквы — по одной букве в ячейку.
2. Слова в кроссворде читаются либо сверху вниз, либо слева направо.
3. Ячейки, выше первого и ниже последнего символов в вертикально расположенном слове, и ячейки, левее первого и правее последнего символа в горизонтально расположенном слове, должны быть пустыми, т.е. не содержать буквы других слов.
4. Ячейки, выше и ниже ячеек горизонтального слова не должны принадлежать другому горизонтальному слову.
5. Ячейки, левее и правее ячеек вертикального слова не должны принадлежать другому вертикальному слову.
6. Каждое слово должно содержать хотя бы одну ячейку, принадлежащую сразу двум словам. Исключение: кроссворды, состоящие из одного слова.

## **Описание метода поиска с возвратом**

Поиск с возвратом — общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов в некотором множестве.

Поиск с возвратом практически одновременно и независимо был изобретен многими исследователями ещё до его формального описания американским математиком Дерриком Генри Лемером в 1950г.

Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше. Данный алгоритм позволяет найти все решения поставленной задачи, если они существуют.

Однако, при построении кроссворда достаточно лишь одного решения. Также перебор всех вариантов кроссворда займёт много большие объёмы ресурсов, чем поиск первого решения. Потому выходные данные программы ограничатся лишь одним кроссвордом.

## **Описание работы программы**

Класс CW хранит в себе:

1. Количество слов в кроссворде на данный момент;
2. Координаты условного «центра» кроссворда;
3. Динамический массив структур, каждая из которых содержит информацию о конкретном слове: координаты первой буквы, расположение слова (горизонтальное / вертикальное), само слово.

Класс CW обладает следующими методами:

1. AddWord — добавляет информацию о слове в массив (и перевыделяет память при необходимости);
2. DeleteWord — удаляет информацию о последнем добавленном слове;
3. Distance — возвращает расстояние от конкретной точки до условного «центра» кроссворда.

Буквы в кроссворде располагаются в клетках, которые описаны классом Cell. Каждый объект класса содержит следующие поля:

wchar\_t symbol;

unsigned canh : 2;

unsigned canv : 2;

signed cansmf : 4;

Поле symbol содержит символ данной ячейки. Битовые поля содержат информацию о том, может ли эта ячейка быть использована для буквы другого слова.

Объекты класса Cell сгруппированы в аналог координатной плоскости, позволяющий обратиться к любой клетке на плоскости. Координатная плоскость реализована при помощи трёхмерного динамического массива, первая координата которого обозначает четверть плоскости, а остальные две являются аналогами координат обычной плоскости.

Конструктор класса DynArr (см. Приложение 1) выделяет память под трёхмерный массив объектов класса Cell. Метод cell (см. Приложение 2) класса DynArr позволяет получить доступ к объекту класса Cell, условно расположенному по координатам (a; b). Метод Reallocate класса DynArr вызывается из метода cell, при попытке обращения к ячейке с координатами за нынешней «границей» массива, и перевыделяет память для динамического массива, увеличивая диапазон значений, к которым может обратиться метод cell.

При работе с динамический массивом использованы функции из библиотеки malloc.h т.к. при перевыделении памяти средствами операторов new-delete необходимо:

1. Выделить другой участок памяти нового размера;
2. Скопировать уже имеющиеся данные в новый участок памяти;
3. И только затем освободить участок памяти, размер которого хотели изменить.

Однако функция realloc может, при определённых условиях, лишь «расширить» имеющийся участок памяти с сохранением имеющихся данных, и, соответственно, необходимость копирования данных отпадёт.

Класс DynArr обладает следующими методами:

1. PosWord (см. Приложение 3) — записывает некоторое слово в трёхмерный массив и вызывает CW::AddWord для этого слова;
2. CanPos — определяет, используя трёхмерный массив объектов класса Cell возможно ли поместить данное слово по данным координатам;
3. Erase — «стирает» из трёхмерного массива слово по данным координатам и вызывает CW::DeleteWord;
4. FindAllPositions — возвращает указатель на структуру AllPositions описывающую все возможные позиции расположения данного слова на данный момент.

Структура AllPositions содержит следующие поля:

int \* x, \* y, \* n, amount;

char \* c;

double \* distance;

Где amount — количество позиций, которые описаны остальными полями структуры.

Функции работы со структурой AllPositions:

1. CreateAP — динамически выделяет память под массив позиций;
2. ReallocateAP — перевыделяет память под массив позиций;
3. QSort — сортирует возможные позиции по значению поля distance;
4. Exclude — удаляет повторяющиеся копии для каждой позиции (такая ситуация возникает, когда слово может быть расположено в пересечении с двумя или более другими словами);
5. SetSize — перевыделяет память под массив позиций. Новый размер для каждого поля равен amount\*sizeof(тип\_соответствующего\_поля).
6. GenerateFirst — динамически выделяет память под массив объектов класса Chosen, исключает из списка слов те, которые являются пустыми (длинна равна нулю), инициализирует поля maxnum (количество всех слов) и param (определяет один из трёх способов предпочтительного расположения слов), вызывает DynArr::PosWord для первого слова, вызывает DynArr::Generate;
7. Generate — возвращает true, если для данного расположения слов (возможно не всех) на плоскости найдена возможность расположить все оставшиеся слова, и false, если такая возможность не найдена.

Класс Chosen обладает всего одним полем — char r. Каждый бит поля r соответствует слову, равен 0, если это слово ещё не использовано в кроссворде, и равен 1 в противном случае. Методы класса Chosen:

1. can — возвращает true, если данный бит равен 0, и false в противном случае;
2. toone, tonul — устанавливают данный бит равным 1 и 0 соответственно.

Хотя хранение слов сразу в двух экземплярах — в трёхмерном массиве Cell и в классе CW — негативно сказывается на потребляемой памяти, добавление класса CW значительно облегчает и ускоряет работу DynArr::FindAllPositions, а координатная плоскость сводит к минимуму необходимые операции для определения, возможно ли поместить данное слово по данным координатам (DynArr::CanPos).

# **Заключение**

Проделав данную работу, я убедился в том, что метод поиска с возвратом позволяет строить кроссворды на большое, по меркам кроссворда, количество слов за маленькие промежутки времени. Работа помогла углубиться в особенности использования динамических структур данных в языке C++. Поставленные цели и задачи выполнены, создано приложение на основе VCL «Генератор кроссвордов». Данное приложение может использоваться:

1. Издателями периодической печати;
2. Учителями и преподавателями ВУЗов, для проверки знаний учащихся, путём составления кроссворда на конкретную тему.

Тема особенно актуальна в настоящее время из-за большого количества людей, увлечённых кроссвордами.

# **Приложение 1.**

DynArr :: DynArr ( ) {

size = 4;

q = ( class Cell \*\*\* ) malloc ( sizeof ( class Cell \*\* ) \* 4 );

for ( int k = 0; k < 4; k++ ) {

q[k] = ( class Cell \*\* ) malloc ( sizeof ( class Cell \* ) \* size );

for ( int i = 0; i < size; i++ ) {

q[k][i] = ( class Cell \* ) malloc ( sizeof ( class Cell ) \* size );

for ( int j = 0; j < size; j++ ) {

q[k][i][j].Cellinit ( );

}

}

}

ChosenWords = NULL;

return;

}

**Приложение 2.**

class Cell \* DynArr :: cell ( int a, int b ) {

while ( Max ( a, b ) >= size ) {

Reallocate ( );

}

int index;

if ( a >= 0 ) {

if ( b >= 0 ) {

return &q[0][a][b];

} else {

return &q[3][a][ - ( b + 1 ) ];

}

} else {

if ( b >= 0 ) {

return &q[1][ - ( a + 1 ) ][b];

} else {

return &q[2][ - ( a + 1 ) ][ - ( b + 1 ) ];

}

}

}

**Приложение 3.**

void DynArr :: PosWord ( int a, int b, wchar\_t \* w, char c, int n ) {

class Cell \* cl;

int len = wcslen ( w );

crossw.AddWord ( a - ( ( c == 'h' ) ? n : 0 ) , b + ( ( c == 'v' ) ? n : 0 ) , w, c );

if ( c == 'h' ) {

for ( int i = a - n; i < a + len - n; i++ ) {

cell ( i, b - 1 ) -> NoH ( );

cl = cell ( i, b );

cl -> SetSH ( w[i - a + n] );

cell ( i, b + 1 ) -> NoH ( );

}

cell ( a - 1 - n, b ) -> Nothing ( );

cell ( a + len - n, b ) -> Nothing ( );

} else {

for ( int i = b + n; i > b - len + n; i-- ) {

cell ( a - 1, i ) -> NoV ( );

cl = cell ( a, i );

cl -> SetSV ( w[b - i + n] );

cell ( a + 1, i ) -> NoV ( );

}

cell ( a, b + 1 + n ) -> Nothing ( );

cell ( a, b - len + n ) -> Nothing ( );

}

return;

}