Crosswords Puzzle

WORK OVERVIEW

The objective of this practical work is to develop two programs, one to create a crosswords puzzle and save it, in a text file, the other to load a puzzle and let the user solve it. Both programs need a synonyms dictionary file that is provided. All the following specifications should be read before starting the development .

LEARNING OBJETIVES

Students are expected to practice the development of programs in C++ language, following the "object-oriented" programming paradigm, selecting suitable data structures and algorithms for implementing the programs and taking advantage of the data structures and algorithms available in the C++ Standard Template Library (STL).

PROGRAM SPECIFICATION

Program 1 – Crosswords maker

This program, named **cwcreator**, will be used to create a crosswords puzzle. It must do the following:

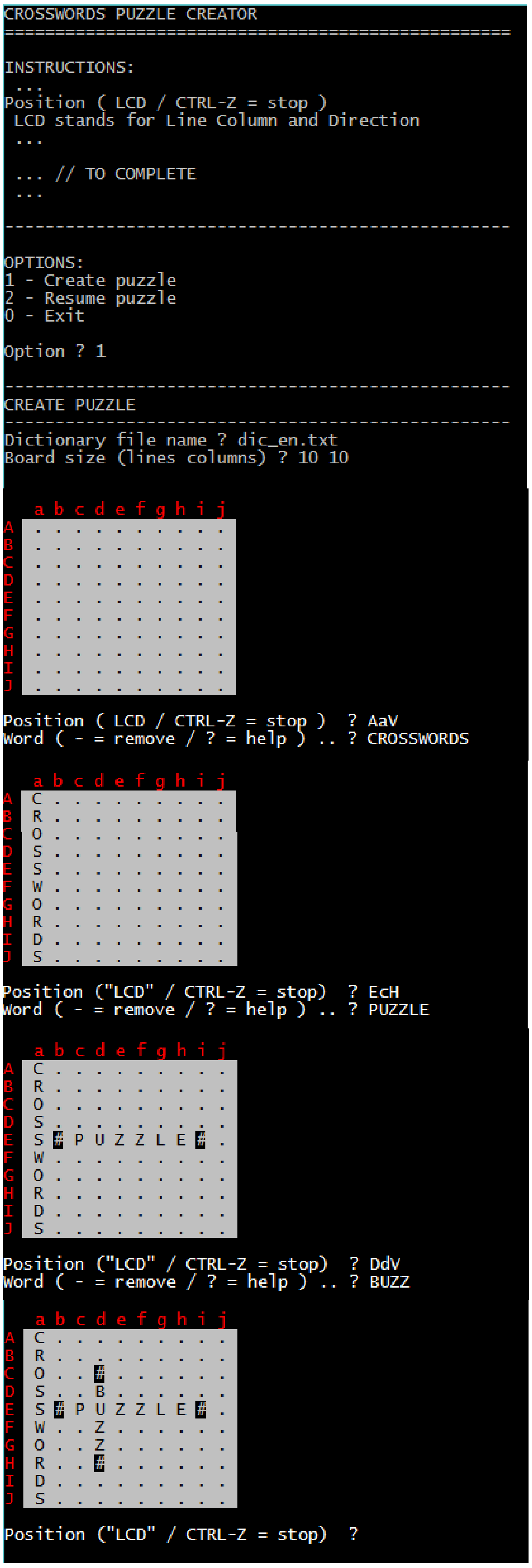
- Read the contents of the dictionary, stored in a text file, isolating all the words and their synonyms.
- Allow the user to choose the size of the board (in vertical and horizontal directions).
- Show the user an empty board.
- Allow the user to insert words, vertically or horizontally, from a given starting cell, verifying namely that: the selected word is valid, that is, it belongs to the dictionary; the word fits in the space between the starting cell and the end of the line/column (depending on the selected word direction); the word has not yet been placed on the board; the word matches with letters from other words that have been placed previously on the board.
- Help the user by showing him/her a list of words that can be added, taking into account the current contents of the board.
- Update and show the contents of the board after each chosen word.
- Repeat the word selection and placement process until the board is full of words or black cells. Notice that the board may contain some black cells before and/or after each word.
- Allow the user to remove a previously placed word.
- Allow to save an unfinished puzzle into a file and reload it later to continue creating the puzzle.
- When the board is full, do a final checking to ensure that all the words that constitute the puzzle are valid.
- Store the resulting board in a text file. The name of the file must have a name in the format **bXXX.txt**, where **XXX** represents the number of the board, always with 3 digits, starting in "001".

This program must implement at least two classes: **Board** and **Dictionary**. The **Board** class should have, among others, functions to build, update and show the contents of the board. The **Dictionary** class should have functions to load the dictionary from the dictionary file, verify that a word is valid, search for words that match some letters already placed on the board (remember the wildcard matching function from practical work number 1) or any others that you find useful. A class **Puzzle** might also be helpful.

Some extracts of a dictionary file and of the resulting board file are shown in annexes A and B, respectively. All the board files must have the exemplified structure: all the letters are represented in uppercase and the black cells are represented by an hashtag character ('#').

The program must run in "console mode" and have a simple interface, like the one shown next.

Example of the initial execution of the program (note that the interface is not yet completely developped):



```
CROSSWORDS PUZZLE CREATOR
==================================================

INSTRUCTIONS:
 ...
Position ( LCD / CTRL-Z = stop )
 LCD stands for Line Column and Direction
 ...

 ... // TO COMPLETE
 ...

--------------------------------------------------

OPTIONS:
1 - Create puzzle
2 - Resume puzzle
0 - Exit

Option ? 1

--------------------------------------------------
CREATE PUZZLE
--------------------------------------------------
Dictionary file name ? dic_en.txt
Board size (lines columns) ? 10 10


   a b c d e f g h i j
A  . . . . . . . . . .
B  . . . . . . . . . .
C  . . . . . . . . . .
D  . . . . . . . . . .
E  . . . . . . . . . .
F  . . . . . . . . . .
G  . . . . . . . . . .
H  . . . . . . . . . .
I  . . . . . . . . . .
J  . . . . . . . . . .

Position ( LCD / CTRL-Z = stop )  ? AaV
Word ( - = remove / ? = help ) .. ? CROSSWORDS

   a b c d e f g h i j
A  C . . . . . . . . .
B  R . . . . . . . . .
C  O . . . . . . . . .
D  S . . . . . . . . .
E  S . . . . . . . . .
F  W . . . . . . . . .
G  O . . . . . . . . .
H  R . . . . . . . . .
I  D . . . . . . . . .
J  S . . . . . . . . .

Position ("LCD" / CTRL-Z = stop)  ? EcH
Word ( - = remove / ? = help ) .. ? PUZZLE

   a b c d e f g h i j
A  C . . . . . . . . .
B  R . . . . . . . . .
C  O . . . . . . . . .
D  S . . . . . . . . .
E  S # P U Z Z L E # .
F  W . . . . . . . . .
G  O . . . . . . . . .
H  R . . . . . . . . .
I  D . . . . . . . . .
J  S . . . . . . . . .

Position ("LCD" / CTRL-Z = stop)  ? DdV
Word ( - = remove / ? = help ) .. ? BUZZ

   a b c d e f g h i j
A  C . . . . . . . . .
B  R . . . . . . . . .
C  O . . # . . . . . .
D  S . . B . . . . . .
E  S # P U Z Z L E # .
F  W . . Z . . . . . .
G  O . . Z . . . . . .
H  R . . # . . . . . .
I  D . . . . . . . . .
J  S . . . . . . . . .

Position ("LCD" / CTRL-Z = stop)  ?
```

Additional development specifications/notes:

- The starting cell (position) of a word is indicated using an uppercase letter that indicates the line and a lowercase letter that indicates the column. These 2 letter must be followed by a letter that indicates the direction of the word: 'H' for horizontal or 'V' for vertical.
- The words to be placed on the board may be input in uppercase or lowercase but they always shown in uppercase.
- The program must be robust against invalid inputs.
- When it is not possible to place a word in a chosen position an adequate error message must be presented to the user.
- The user may stop the creation by typing CTRL-Z (CTRL-D, in Linux) in answer to the "Position?" question. Then he/she must be asked whether he/she wants to save the board and resume later or to finish. In this last case, if all the words are valid, all the cells that have not yet been filled (do not contain letters and are not black) must be put black.
- In all other situations, typing CTRL-Z (or CTRL-D) should have no effect on the running of the program.
- The user may ask for help, that is, ask for a list of possible words that fit into the specified position, by typing "?" in answer to the "Word?" question.
- It must be possible to remove some words previously placed on the board, by typing "-" in answer to the "Word?" question (after specifying the coordinates of the starting cell).
- You are free to add other options/specifications. For example, it may happen that some "words" are automatically formed when other words are placed on the board. One could add the possibility of checking if an "automatically formed word" is valid by specifying the "word" position and typing "+" in answer to the "Word?" question. You must describe all the added specifications in a text file that you must send together with the code.
- The classes **Board** and **Dictionary** must be reused in the second program: Crosswords Player.
- To develop this project you won't be starting from scratch. On top of all the functions you have developed for the first project (and should reuse as possible), you can also use code provided to you to display colored characters on the console (see annex C).
- Do not clear the screen between the operations executed by the program, to allow scrolling back (use the screen capture presented in the previous page as an example).

Program 2 – Crosswords player

This program, named **cwplayer**, must allow the user to solve a previously created crosswords puzzle, produced by program **cwcreator** and stored in a text file as specified before. It must do the following:

- Read the contents of the dictionary, stored in a text file, isolating all the words and their synonyms (reuse the code from **cwcreator**).
- Read a puzzle, from a text file (reuse the code from **cwcreator**). NOTE: see in the specification of **cwcreator** the name to give to the board files.
- Read the name of the player.
- Show the user an "empty board" (a "grid" without words; only with white and black cells).
- Show an initial list of clues for the puzzle words; the position of each word is indicated by 2 letters, as in **cwcreator**. The clue is a synonym for each word to be guessed. The synonym must be chosen randomly from the set of synonyms available in the dictionary. The clues must be shown to the player grouped in 2 sets: HORIZONTAL and VERTICAL, for the horizontal and vertical words.
- Allow the user to solve the puzzle by placing on the board the word that corresponds to each clue and meets a number of constraints: a length constraint, provided by the grid, and constraints on its letters, provided by the words which it overlaps in the grid. The verification that the placed word is correct should not be done immediately; it will be done only when the board is full.
- As it happens with **cwcreator** program, the player must have the possibility to remove a word or to place a word over a previously placed word.
- The user may also ask for help, by indicating the position of a word on the board and typing "?" in answer to the "Word ?" question. In this case, the program must show an alternative clue (if the word in question has more than one synonym).

- When the board is full, do a final checking to ensure that all the words placed on the board are valid.
- If the puzzle was solved successfully, save the player data to a file, as specified in the following.

This program must implement at least the following classes: **Board** and **Dictionary** and **Player.** The code previosuly developed for classes **Board** and **Dictionary** should be reused. However, it may be necessary to add some methods that you forgot to include, for example, to show an "empty board"/"grid". The **Player** class should have, at least, the following attributes: the name of the player, the total time (in seconds) he/she took to solve the puzzle and the number of alternative clues that he/she asked. Whenever a player manages to solve a puzzle, this information must be appended to a file named **bXXX_p.txt** where **XXX** represents the number of the board, always with 3 digits, starting in "001". The information about each player must be written in a single line of text. A class **Puzzle** might also be helpful.

You are free to add other options/specifications. All the additional specifications must be described in a text file that you must send together with the code.

The program must run in "console mode" and have a simple interface. You are free to choose the interface. Do not clear the screen during program execution, to allow scrolling back (use the the screen capture presented in the previous page as an example).

SUGGESTION: do not build large puzzle boards, unless you are very proficient in English.

PROGRAM DEVELOPMENT

NOTE: anything that is not specified in this project specification may be specified by the members of the group and the additional specifications must be described in the **ReadMe.txt** file (see notes on the work submission, below).

Code writing

When writing the program code you should take into account the suggestions given in class, specially the ones concerning the following issues:

- Adequate choice of identifiers of types, variables and functions.
- Code commenting.
- Adequate choice of the data structures to represent the data manipulated by the program.
- Modular structure of the code.
- Separation, as much as possible, of data processing from program input/output.
- Code robustness. Precautions should be taken in order to prevent the programs to stop working due to incorrect input by the user, specially values outside the allowed ranges, non existent files, and so on.
- Code efficiency; try to write efficient code.

WORK SUBMISSION

- Create a folder named **TxGyy**, in which **x** represents the class number (Portuguese "turma") and **yy** represents the group number (with 2 digits), for example, **T5G07**, for the group 7 from class ("turma") 5. Create three subfolders, one named **P1**, one named **P2,** and another named **Puzzles**. Copy to subfolder **P1** the source code (only the files with extension **.cpp** and **.h**, if existing) of Program 1 (**cwcreator**), to subfolder **P2** the source code of Program 2 (**cwplayer**). In subfolder **Puzzles** put: two puzzles that you have created using the provided dictionary, one finished puzzle (name it **b001.txt**) and another not yet finished (**b002.txt**); the file (**b001_s.txt**) that contains the data about the players that managed to solve the finished puzzle. Include also a file, **ReadMe.txt** (in simple text format), indicating the development state of the programs, that is, if all the objectives were accomplished or, otherwise, which ones were not achieved, and also what improvements were made, if any.

- Compress the content of the folder **TxGyy** into a file named **TxGyy.zip** or **TxGyy.rar**, depending on the compression tool that you use, and upload that file in the FEUP Moodle's page of the course. Alternative ways of delivering the work will not be accepted.

- Deadline for submitting the work: **14th/May/2018 (at 09:00h)**.

ANNEX A

Sample of the provided synonyms dictionary (synonyms.txt)

Notes:
1- Although, in the sample below, some lists of synonyms occupy more than one line, in the provided file they are in the same text line.
2- The ellipsis ("...") indicate ommited text.

```
Aback: backwards, rearwards, aft, abaft, astern, behind, back
Abandon: leave, forsake, desert, renounce, cease, relinquish, discontinue, castoff, resign, retire, quit, forego, forswear, depart
from, vacate, surrender, abjure, repudiate
Abandoned: profligate, wicked, vicious, unprincipled, reprobate, incorrigible, sinful, graceless, demoralized, dissolute, depraved,
bad, licentious, corrupt
Abase: degrade, disgrace, bring low, reduce, humble, demean, stoop, humiliate, depress, lower, sink, dishonor
Abasement: degradation, depression, disgrace, humiliation, abjection, dishonor, shame
Abash: confound, confuse, discompose, bewilder, daunt, cow, humble, disconcert, dishearten, motility, shame, humiliate
Abate: terminate, remove, suppress, lower, reduce, mitigate, diminish, moderate, lessen, subside, decrease
Abbreviate: shorten, reduce, abridge, contract, curtail, epitomize, condense, prune, compress
Abbreviation: abridgment, reduction, contraction, curtailment, abstract, summary, epitome, condensation, compression
Abderite: cynic, sardonic, derisive, sarcastic, jocular, flippant, nonchalant, sportive, sprightful, buoyant
Abdicate: abandon, relinquish, resign, surrender, vacate
Abdomen: belly, paunch
Abduction: abstraction, subtraction, deprivation, ablution, rape, seizure, appropriation
Aberrant: erratic, devious, divergent, incontinuous, desultory, disconnected, wandering, idiotic, inconsistent, inconsecutive,
abnormal, exceptional
Aberration: wandering, divergence, deviation, desultoriness, rambling, disconnectedness, hallucination, inconsecutiveness, idiocy,
insanity, exception, abnormity

...

Wrong: unfit, unsuitable, improper, mistaken, incorrect, erroneous, unjust, illegal, inequitable, immoral, injurious, awry
Wrongdoer: delinquent, sinner, culprit, offender, evil-doer, malefactor
Wrong: evil-doing, wickedness, injustice, crime
Wrongful: unfair, unjust, dishonest, wrong, iniquitous
Wrought: performed, effected, executed, done, produced, manufactured
Wry: long, hanker, crave, covet, desire
Yearly: annually, year by year, every year, per annum
Yet: besides, nevertheless, notwithstanding, however, still, eventually, ultimately, at last, so far, thus far
Yield: furnish, produce, afford, bear, render, relinquish, give in, let go, forego, accede, acquiesce, resign, surrender, concede,
allow, grant, submit, succumb, comply, consent, agree
Yielding: conceding, producing, surrendering, supple, pliant, submissive, unresisting
Yoke: couple, conjoin, connect, link, enslave, subjugate
Youth: youngster, young person, boy, lad, minority, adolescence, juvenility
Youthful: ardor, interest energy, eagerness, engagedness, heartiness,
Zealot: partisan, bigot, enthusiast, fanatic, devotee, visionary
Zealous: ardent, anxious, earnest, enthusiastic, fervid, eager, steadfast
Zenith: height, highest point, pinnacle, acme, summit, culmination, maximum
Zephyr: west wind, mild breeze, gentle wind
Zero: naught, cipher, nothing
```

ANNEX B

Example of files that represent boards

Notes:
1- The first line of the board file must contain the name of the dictionary file that was used to verify the words. The second line is empty. The following lines show the board. After the board, there is another empty line and the list of words and their positions. The synonyms, to be presented to the player, are not stored in this file. They will be chosen when the board will be loaded by the Crosswords Player program.
2- The following boards were not verified using the provided dictionary.

BOARD NOT FINISHED:

```
mydictionary.txt

D A D # . . . . D  ← updated on 2018-05-04
O # E A S T # . A
W # A . . . . . Y
N E R F # . . . #
# A # A R K # . #
# S . . . . Y . C
M E S H # . A . A
A # . . . . R . R
N . . . . . D . D

AaH DAD            ← updated on 2018-05-04
AaV DOWN
GaV MAN
AbV DEAR
DaH NERF
DbV EASE
GaH MESH
BcH EAST
EdH ARK
FgV YARD
AiV DAY
FiV CARD
```

BOARD FINISHED:

```
mydictionary.txt

D A D # # S E N D
O # E A S T # # A
W # A # # I T S Y
N E R F # N # T #
# A # A R K # U #
# S # T # S Y N C
M E S H # # A # A
A # # E V E R # R
N E A R # # D # D

AaH DAD
AfH SEND
BcH EAST
CfH ITSY
DaH NERF
EdH ARK
FfH SYNC
GaH MESH
HdH EVER
IaH NEAR
AaV DOWN
GaV MAN
DbV EASE
AcV DEAR
DdV FATHER
AfV STINKS
FgV YARD
AiV DAY
FiV CARD
```

ANNEX C

Example of a program that prints colored characters on the console

MS-Windows/DOS version

```
// PROG - MIEIC
// JAS
// Example of a program that prints colored characters on the console (in text mode)

#include <iostream>
#include <ctime>
#include <cstdlib>
#include <windows.h>

using namespace std;

//==========================================================================================
//COLOR CODES: (alternative: use symbolic const's)
#define BLACK 0
#define BLUE 1
#define GREEN 2
#define CYAN 3
#define RED 4
#define MAGENTA 5
#define BROWN 6
#define LIGHTGRAY 7
#define DARKGRAY 8
#define LIGHTBLUE 9
#define LIGHTGREEN 10
#define LIGHTCYAN 11
#define LIGHTRED 12
#define LIGHTMAGENTA 13
#define YELLOW 14
#define WHITE 15

//==========================================================================================

void clrscr(void)
{
  COORD coordScreen = { 0, 0 };  // upper left corner
  DWORD cCharsWritten;
  DWORD dwConSize;
  HANDLE hCon = GetStdHandle(STD_OUTPUT_HANDLE);
  CONSOLE_SCREEN_BUFFER_INFO  csbi;

  GetConsoleScreenBufferInfo(hCon, &csbi);
  dwConSize = csbi.dwSize.X * csbi.dwSize.Y;

  // fill with spaces
  FillConsoleOutputCharacter(hCon, TEXT(' '), dwConSize, coordScreen, &cCharsWritten);
  GetConsoleScreenBufferInfo(hCon, &csbi);
  FillConsoleOutputAttribute(hCon, csbi.wAttributes, dwConSize, coordScreen, &cCharsWritten);

  // cursor to upper left corner
  SetConsoleCursorPosition(hCon, coordScreen);
}

//==========================================================================================

// Position the cursor at column 'x', line 'y'
// The coodinates of upper left corner of the screen are (x,y)=(0,0)

void gotoxy(int x, int y)
{
  COORD coord;
  coord.X = x;
  coord.Y = y;
  SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

//==========================================================================================

// Set text color

void setcolor(unsigned int color)
{
  HANDLE hcon = GetStdHandle(STD_OUTPUT_HANDLE);
  SetConsoleTextAttribute(hcon, color);
}

//==========================================================================================

// Set text color & background

void setcolor(unsigned int color, unsigned int background_color)
{
  HANDLE hCon = GetStdHandle(STD_OUTPUT_HANDLE);
  if (background_color == BLACK)
    SetConsoleTextAttribute(hCon, color);
  else
    SetConsoleTextAttribute(hCon, color | BACKGROUND_BLUE | BACKGROUND_GREEN |
    BACKGROUND_RED);
}

//==========================================================================================

// Fill the screen with colored numbers

int main()
{
  clrscr();

  srand((unsigned int)time(NULL));

  for (int x = 0; x < 80; x++)
    for (int y = 0; y < 24; y++)
    {
      gotoxy(x, y);
      if (rand() % 2 == 0)
        setcolor(x % 15 + 1);
      else
        setcolor(y % 15 + 1, rand() % 2);
      cout << x % 10;
    }
}
```

Linux version (using ANSI escape sequences)

```
// PROG - MIEIC
// JAS
// Example of a program that prints colored characters on the console (in text mode)
// LINUX version, using ANSI escape sequences

#include <iostream>
#include <string>
#include <sstream>
#include <unistd.h>
using namespace std;
//==========================================================================================
// TEXT COLOR CODES
#define NO_COLOR      "\033[0m"
#define BLACK         "\033[0;30m"
#define RED           "\033[0;31m"
#define GREEN         "\033[0;32m"
#define BROWN         "\033[0;33m"
#define BLUE          "\033[0;34m"
#define MAGENTA       "\033[0;35m"
#define CYAN          "\033[0;36m"
#define LIGHTGRAY     "\033[0;37m"
#define DARKGRAY      "\033[1;30m"
#define LIGHTRED      "\033[1;31m"
#define LIGHTGREEN    "\033[1;32m"
#define YELLOW        "\033[1;33m"
#define LIGHTBLUE     "\033[1;34m"
#define LIGHTMAGENTA  "\033[1;35m"
#define LIGHTCYAN     "\033[1;36m"
#define WHITE         "\033[1;37m"
// BACKGROUND COLOR CODES
#define BLACK_B       "\033[0;40m"
#define RED_B         "\033[0;41m"
#define GREEN_B       "\033[0;42m"
#define YELLOW_B      "\033[0;43m"
#define BLUE_B        "\033[0;44m"
#define MAGENTA_B     "\033[0;45m"
#define CYAN_B        "\033[0;46m"
#define WHITE_B       "\033[1;47m"
//==========================================================================================
// Position the cursor at column 'x', line 'y'
// The coodinates of upper left corner of the screen are (x,y)=(0,0)
void gotoxy(int x, int y)
{
  ostringstream oss;
  oss << "\033[" << y << ";" << x << "H";
  cout << oss.str();
}
//==========================================================================================
// Clear the screen
void clrscr(void)
{
  cout << "\033[2J";
  gotoxy(0, 0);
}
//==========================================================================================
// Set text color
void setcolor(string color)
{
  cout << color;
}
//==========================================================================================
// Set text color & background
void setcolor(string color, string background_color)
{
  cout << color << background_color;
}
//==========================================================================================
// Testing program
int main()
{
  clrscr();
  cout << RED << "Text in RED" << NO_COLOR << endl;
  cout << LIGHTRED << "Text in LIGHTRED" << NO_COLOR << endl;
  cout << BLUE << "Text in BLUE" << NO_COLOR << endl;
  cout << GREEN << "Text in GREEN" << NO_COLOR << endl;
  cout << RED << WHITE_B << "Text in RED on WHITE background" << NO_COLOR << endl;
  cout << RED << BLACK_B << "Text in RED on BLACK background" << NO_COLOR << endl;
  cout << "\nPress <enter> to continue ..."; cin.get();

  cout << "Cursor is going to move to (20,3). Press <enter> to continue ..."; cin.get();
  gotoxy(20, 3);

  cout << "Screen is going to be cleaned. Press <enter> to continue ..."; cin.get();
  clrscr();

  // alternatively ...
  setcolor(LIGHTBLUE, WHITE_B); // OR cout << LIGHTBLUE << WHITE_B;
  cout << "From now on\n";
  cout << "everything is written LIGHTBLUE\n";
  cout << "on WHITE background\n";
  setcolor(NO_COLOR); // OR cout << NO_COLOR;
  cout << "End of program\n";

}
```

Linux version (using NCURSES library)

```cpp
// PROG - MIEIC
// ZP
// Example of a program that prints colored characters on the console (in text mode)

#include <iostream>
#include <ctime>
#include <ncurses.h>

using namespace std;

/**
 * Called after using colored text
 */
void finish()
{
  resetterm();
  cout << std::endl;
}

/**
 * This method must be called before using colored text
 */
void init()
{
  short f, b;

  initscr();
  start_color();
  use_default_colors();

  // initialize default colors
  // COLORS is defined by Curses as the total number of colors
  for (f = 0; f < COLORS; ++f)
  {
    init_pair(f, f, -1);
    for (b = 0; b < COLORS; ++b)
      init_pair((short)(f * COLORS + b), f, b);
  }
  atexit(finish);
}

/**
 * Clear the entire screen
 */
void clrscr()
{
  clear();
  refresh();
}

/**
 * Output colored text to screen
 * @param text The text to output
 * @param x column in the screen
 * @param y row in the screen
 * @param fg_color Foreground color
 * @param bg_color Background color
 */
void write_text(const string& text, unsigned int x, unsigned int y, int fg_color, int bg_color = -1) {
  move(y, x);

  if (bg_color == -1)
    attron(COLOR_PAIR(fg_color));
  else
    attron(COLOR_PAIR(fg_color * COLORS + bg_color));

  addstr(text.c_str());
  refresh();
}

// Fill the screen with colored numbers
int main()
{
  srand((unsigned int)time(nullptr));
  char more;

  init();

  do
  {
    clrscr();
    for (unsigned int x = 0; x < 80; x++)
      for (unsigned int y = 0; y < 24; y++)
      {
        // COLOR_GREEN, COLOR_RED, ... are defined by Curses
        // taking values from 0 to COLORS (also defined there)
        write_text(to_string(x % 10), x, y, rand() % COLORS, COLOR_GREEN);
      }
    move(25, 0);
    refresh();
    cout << "more? (Y/N)";
    cin >> more;
  } while (toupper(more) == 'Y');

}
```

- Colors defined in **ncurses.h**: COLOR_BLACK, COLOR_RED, COLOR_GREEN, COLOR_YELLOW, COLOR_BLUE, COLOR_MAGENTA, COLOR_CYAN, COLOR_WHITE. Reference: http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/color.html