Instrukcja projektowa; projekt numer 1

Podstawy Programowania 2014/15, kierunek Informatyka

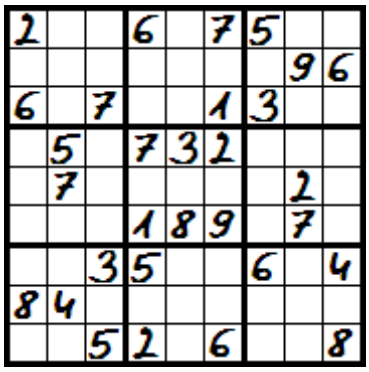
autor: Krzysztof Bruniecki[1]

Projekt SUDOKU

Cel

Celem projektu jest realizacja konsolowego programu służącego do "ręcznego" rozwiązywania łamigłówki SUDOKU.

SUDOKU – łamigłówka, której celem jest wypełnienie diagramu 9 × 9 w taki sposób, aby w każdym wierszu, w każdej kolumnie i w każdym z dziewięciu pogrubionych kwadratów 3 × 3 (zwanych „blokami” lub „podkwadratami”) znalazło się po jednej cyfrze od 1 do 9 [Wiki-Sudoku]



Ogólne wytyczne

Projekt powinien być napisany w języku C z możliwością stosowania podstawowych elementów języka C++ (podobnie jak podczas laboratoriów). Projekt może być pisany w sposób obiektowy, ale całkowicie **zabronione jest użycie biblioteki STL.**

Do obsługi formatu XML **nie można** stosować specjalistycznych bibliotek (parserów) XML.

Za projekt można uzyskać 0-18 punktów. 15 punktów stanowi tzw. 100% pozostałe 3 punkty to wymagania ponadprogramowe ("z gwiazdką").

Realizacja wymagań obowiązkowych konieczna jest aby projekt był w ogóle oceniany.

[1] Uwaga: W razie niejasności lub niejednoznaczności w poniższym opisie proszę kontaktować sie z autorem instrukcji; pokój 645EA; konsultacje odbywają się w poniedziałki 13:15-15:00

Program powinien być napisany z użyciem szablonu dostępnego w materiałach. Szablon umożliwia uzyskanie zaawansowanych możliwości w zakresie obsługi konsoli w systemie Windows.

Uwaga!. Udostępniona biblioteka działa w systemie Windows i nie ma możliwości łatwego przeniesienia jej do innych systemów. Osoby, które piszą program w systemie Linux powinny wykorzystać bibliotekę **ncurses**.

Podczas oddawania (aby usprawnić oddawanie) należy przygotować planszę przykładową (częściowo wypełnioną). Student oddający powinien móc wczytać ją z pliku, albo powinna wyświetlić się w programie po wybraniu klawisza "i" (jeśli nie zrealizowano wczytywania z pliku). Plansza przykładowa powinna być taka jak ta na stronie tytułowej niniejszej instrukcji.

Obsługa programu

Program powinien wykorzystywać klawiaturę w następujący sposób:

strzałki - przesuwanie kursora, próba wyjścia za krawędź powinna być odrzucana (np. strzałka w górę przy górnej krawędzi nie przesuwa kursora)

1..9 - wstawienie liczby i nadpisanie liczby
backspace lub del - usunięcie cyfry
u - undo
r - redo
l - lista możliwych cyfr (mała litera L)
p - podpowiedz
s - zapisanie do pliku
o - odczytanie pliku
k - wejście w tryb edycji komentarza

Wymagania obowiązkowe (5pkt)

1. **(1pkt)** Wyświetlanie tekstowe planszy wraz z krawędziami z rozróżnieniem krawędzi grubych i cienkich. Rozmieszczenie elementów na ekranie (plansza, pomoc, licznik ruchów itp.) powinno być łatwo modyfikowalne z poziomu kodu (użycie stałych). Proszę spodziewać się podczas odpowiedzi z kodu np. prośby o przesunięcie planszy o parę znaków w dowolną stronę czy zamienienia miejscami planszy i pomocy).

2. **(1pkt)** Możliwość dodawania liczb w wolne pola aż do całkowitego wypełnienia planszy.

3. **(1pkt)** Możliwość usunięcia wprowadzonej liczby oraz możliwość zmiany (nadpisania) danej liczby.

4. **(1pkt)** Sprawdzanie i blokada wprowadzenia gdy występuje konflikt (wiersz, kolumna lub kwadrat) przy wprowadzaniu danej liczby.

5. **(1pkt)** Użytkowanie programu powinno być intuicyjne i jeśli potrzeba podstawowe komendy powinny być podpowiadane (wyświetlony help).

Wymagania nieobowiązkowe (10 pkt)

1. **(2pkt)** Podpowiedź
   a. Wskazanie możliwych (nie powodujących konfliktu w wierszu kolumnie i podkwadracie) liczb dla wybranego przez użytkownika pola (klawisz l).
   b. Znalezienie pola dla którego podpowiedź z punktu a jest jednoznaczna i propozycja wstawienia tam tej liczby (użytkownik potwierdza chęć skorzystania z podpowiedzi) (klawisz p).

2. **(1pkt)** Możliwość zapisu i odczytu plików z łamigłówką (nazwa pliku powinna być możliwa do podania przez użytkownika, a nie zapisana na stałe w kodzie źródłowym). Pliki powinny być tekstowe i możliwe do edycji w notatniku.

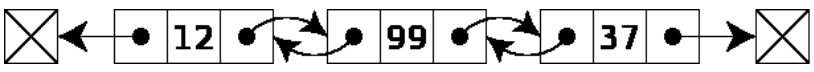
3. **(1pkt)** Realizacja funkcji UNDO (cofnięcie ruchu) oraz REDO (ponowienie po cofnięciu) z zapamiętywaniem co najmniej 10 ruchów wstecz i do przodu. Każda pojedyncza zmiana dowolnej liczby w łamigłówce (tj. wstawienie, zmiana, usunięcie) powinna być na liście UNDO-REDO.

4. **(0,5pkt)** Liczenie ilości wykonanych ruchów (wliczając w to operacie UNDO i REDO) oraz liczenie czasu gry od momentu wystartowania danej planszy. Wartości te powinny być wyświetlone na ekranie. Czas nie musi odświeżać się na bieżąco (należy odświeżać go przynajmniej po każdym ruchu).

5. **(0,5pkt)** Możliwość tymczasowego oznaczenia (np. innym kolorem lub poprzez dodanie znaku *) wszystkich pól zawierających dowolną wybraną przez użytkownika liczbę.

6. **(1pkt)** Dynamiczna alokacja pamięci w celu zapewnienia "potencjalnie" nieskończonej listy dla UNDO i REDO. Z każdym ruchem rośnie historia ruchów. W związku z tym, za każdym razem gdy historia rośnie powinna być alokowana dynamicznie (malloc lub new) pamięć w ilości potrzebnej do zapisu całej historii (tj. dotychczasowa historia + nowy ruch). Proszę zwrócić uwagę, że faktycznie głębokość historii może być nieskończona - np. użytkownik może wpisywać w pole na przemian dwie pasujące liczby w nieskończoność.

7. **(1pkt)** Struktury danych potrzebne dla UNDO i REDO powinny być zbudowane z użyciem wskaźników tj. poszczególne stany łamigłówki powinny być powiązane wskaźnikami (dzięki temu można przechodzić pomiędzy sąsiednimi stanami). Struktura danych tego rodzaju nazywa się listą. Poniżej znajduje się schematyczny rysunek listy [Wiki-Lista] (w tym wariancie tzw. listy dwukierunkowej) składającej się z 3 elementów powiązanych wskaźnikami.



Pojedynczy element listy dla celów projektu może być zdefiniowany następująco:

```
struct SudokuState {
// tutaj dane służące do zapisania pojedynczego stanu planszy
struct SudokuState* next;     //wskaźnik na następny stan
struct SudokuState* prev;     //wskaźnik na poprzedni stan
}
```

8. **(1pkt)** Możliwość dodania do 9 "cyfr roboczych" (będących formą komentarza) dla wybranego pola. Zmiany w "cyfrach roboczych" powinny być części listy UNDO-REDO. Każda pojedyncza zmiana dowolnej liczby w komentarzu (tj. wstawienie, zmiana, usunięcie) powinno być częścią listy UNDO-REDO.

9. **(2pkt)** Obsługa odczytu i zapisu do formatu XML zgodnego z zamieszczonym przykładem.

```
<?xml version="1.0"?>
<sudoku active-state="1">
<state nr="1">
        <board>
                2 - - 6 - 7 5 - -
                - - - - - - - 9 6
                6 - 7 - - 1 3 - -
                - 5 - 7 3 2 - - -
                - 7 - - - - - 2 -
                - - - 1 8 9 - 7 -
                - - 3 5 - - 6 - 4
                8 4 - - - - - - -
                - - 5 2 - 6 - - 8
        </board>
        <comments>
        <comment row="3" col="3"> 2 3 4</comment>
        <comment row="7" col="1">1 4</comment>
        </comments>
</state>
<state nr="2">
        <board>
                2 - - 6 - 7 5 - -
                - - - - - - - 9 6
                6 - 7 - - 1 3 - -
                - 5 - 7 3 2 - - -
                - 7 - - - - - 2 -
                - - - 1 8 9 - 7 -
                - - 3 5 - - 6 - 4
                8 4 - - - - - - -
                - - 5 2 - 6 - - -
        </board>
        <comments>
        <comment row="3" col="3"> 2 3 4</comment>
        <comment row="7" col="1">1 4</comment>
        </comments>
</state>
</sudoku>
```

W tym przykładzie zapisane są dwa stany które miały miejsce podczas rozwiązywania układanki Sudoku. W ogólności liczba stanów może być dowolna. Stan o numerze (atrybut "nr") zgodnym z "active-state" powinien wyświetlić się po wczytaniu pliku. Pozostałe stany powinny po wczytaniu zasilić listę UNDO-REDO.

Opis formatu XML

a. Pierwsza linia powinna zawsze mieć postać jak w powyższym przykładzie.
b. W węźle "sudoku" występuje dokładnie 1 atrybut o nazwie "active-state", którego wartość jest liczbą naturalną.
c. Wewnątrz węzła "sudoku" znajduje się co najmniej 1 węzeł "state".
d. Każdy węzeł "state" (reprezentuje stan z historii) zawiera atrybut "nr" o wartości będącej liczbą naturalną. Musi istnieć węzeł "state" o takiej wartości atrybutu "nr",

która występuje w atrybucie "active-state". Stany o atrybucie "nr" mniejszym od aktualnego stanowią kolejne na liście UNDO, natomiast te o atrybucie "nr" większym od aktualnego na liście REDO (atrybut "nr" porządkuje historię zgodnie z relacją <).

e. Każdy węzeł "state" **zawiera dokładnie 1 węzeł** "board", który zawiera 81 rozdzielonych znaków należących do zbioru {-, 1, 2, 3, 4, 5, 6, 7, 8, 9}. Są one rozdzielone dowolnymi białymi znakami. Węzeł "board" zawiera cyfry którymi wypełniona jest łamigłówka Sudoku. Podane są one w kolejności wiersz po wierszu.
f. Węzeł "state" **może zawierać maksymalnie 1 węzeł** "comments", który zawiera dowolną liczbę węzłów "comment".
g. Każdy węzeł "comment" zawiera atrybuty "row" (wiersz) oraz "col" (kolumna), które mają przypisaną wartość będącą liczbą naturalną z przedziału 1-9 wskazującą jednoznacznie pole z łamigłówki.
h. każdy węzeł "comment" zawiera tekst składający się z maksymalnie 9 liczb z zakresu 1-9. Liczby te to wspomniane wcześniej "cyfry robocze" będące formą komentarza.
i. Kolejność występowania podwęzłów w dowolnym węźle może być dowolna.
j. Kolejność występowania atrybutów w węźle może być dowolna.
k. Dowolna sekwencja białych znaków powinna być traktowana jak separator (nie ważne czy jest pojedyncza spacja, dwa tabulatory czy przykładowo "\n\t " - zawsze traktowane jest to po prostu jako separator).
l. Kodowanie dokumentu - ASCII.
m. Nie ma potrzeby implementowawać wszystkich elementów standardu XML, np. komentarzy.

Następujący dokument jest z logicznego punktu widzenia taki sam jak wcześniejszy przykład

```
<?xml version="1.0"?>
<sudoku active-state="1">
<state nr="2">
        <board>
                2 - - 6 - 7 5 - -
                - - - - - - - 9 6
                6 - 7 - - 1 3 - -
                - 5 - 7 3 2 - - -
                - 7 - - - - - 2 -
                - - - 1 8 9 - 7 -
                - - 3 5 - - 6 - 4
                8 4 - - - - - - -
                - - 5 2 - 6 - - -
        </board>
        <comments>
        <comment row="3" col="3"> 2 3 4</comment>
        <comment row="7" col="1">1 4</comment>
        </comments>
</state>

<state nr="1">

        <comments><comment  row="3"  col="3">  2  3  4</comment><comment  col="1"  row="7">1
4</comment></comments>
        <board>
2 - - 6 - 7 5 - -             - - - - - - - 9 6
6 - 7 - - 1 3 - -             - 5 - 7 3 2 - - -
- 7 - - - - - 2 -             - - - 1 8 9 - 7 -
- - 3 5 - - 6 - 4             8 4 - - - - - - -
- - 5 2 - 6 - - 8
        </board>
</state></sudoku>
```

Wymagania z gwiazdką

Te punkty podlegają ocenie pod warunkiem realizacji wszystkich poprzednich). Do zdobycia jest łącznie 3pkt.

1. (1pkt) Zaimplementować podpowiedź na podstawie "zaawansowanej" analizy opartej na następującym spostrzeżeniu [Wiki-Sudoku].
Metoda polega na znajdowaniu miejsca, gdzie w obrębie małego kwadratu 3 × 3 pasuje dana cyfra na zasadzie eliminacji rzędów i kolumn, w których ta cyfra znajduje się w innych kwadratach.

Diagram 1 – cyfrę 4 wpisać można tylko w jedno pole środkowego dolnego kwadratu (oba pozostałe rzędy są już zajęte).

Diagram 2 – bardziej skomplikowany przypadek, znalezienie miejsca dla cyfry 3. Cyfra 3 pasuje w dwa miejsca w środkowym dolnym kwadracie. Pozwala to na wyeliminowanie tego rzędu (cyfra 3 musi znaleźć się w tym rzędzie, niezależnie, czy na polu po lewej czy po prawej), więc w prawym dolnym kwadracie dwa rzędy są zajęte. Jedną kolumnę zajmuje wpisana już cyfra 3, więc pozostaje jedyne pole, gdzie można wpisać cyfrę 3.(to obok 8)

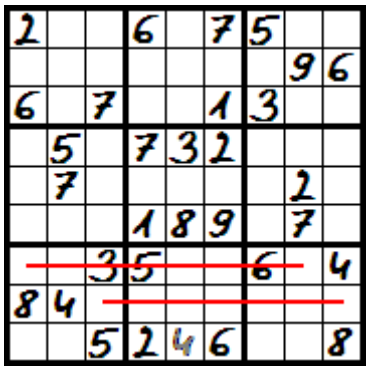


Diagram 1

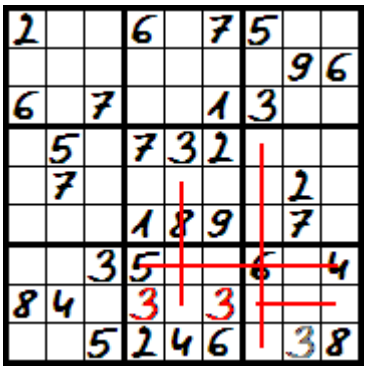


Diagram 2

2. (1pkt) Zaimplementować podpowiedź na podstawie "zaawansowanej" analizy opartej na następującym spostrzeżeniu [Wiki-Sudoku].

Spostrzeżenie polega na dopełnianiu rzędu, kolumny lub kwadratu 3 × 3 cyframi od 1 do 9.

Diagram 3 – w dolnym rzędzie brakuje już tylko dwóch cyfr, Łatwo sprawdzić, że są to 1 i 7. Do drugiego pustego pola od lewej pasuje tylko cyfra 1, ponieważ w tej kolumnie już znajduje się cyfra 7. Cyfra 7 natomiast powinna się znaleźć w pierwszym pustym polu po lewej.

Diagram 4 – w pewnym momencie można dopełnić cały kwadrat, dla przykładu lewy dolny. Cyfra 2 pasuje tylko do środkowej kolumny, cyfra 6 tylko do środkowego rzędu. Do tego, gdzie umiejscowić cyfrę 9, można w tym przypadku dojść na dwa sposoby:

- bo jest to ostatnia cyfra, jaka pozostała do wpisania w tym kwadracie,
- bo nie można tam wpisać ani cyfry 2, ani cyfry 6.

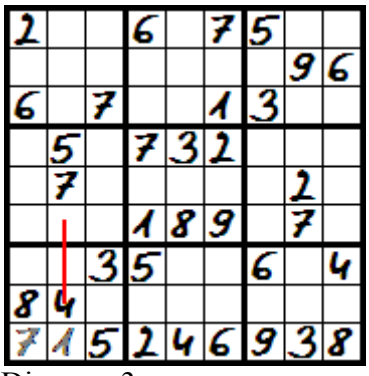


Diagram 3

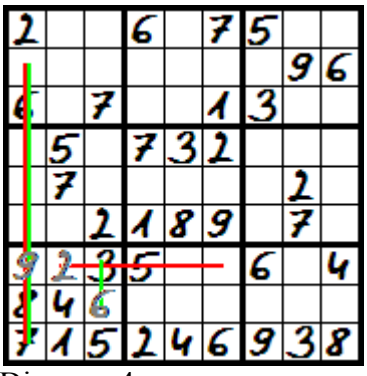


Diagram 4

3. (1pkt) Zaimplementować podpowiedź (lub rozwiązanie końcówki) na podstawie wyczerpującego przeszukiwania zbioru rozwiązań. Program rozwiązujący ma mieć postać trochę lepszą niż naiwne podejście *brute force*. Wymagane podejście, które należy zaimplementować opisane jest w formie przykładu poniżej.

Np. gdy na planszy pozostanie 10 wolnych pól i zgodnie z zasadami w każde z nich można wprowadzić liczbę będącą jedną z 9 możliwości to liczba możliwości wypełnienia planszy wynosi 9^10 = 3 486 784 401. Liczenie wszystkich możliwych rozwiązań i sprawdzenie która z nich nie powoduje żadnego konfliktu byłoby trochę zbyt czasochłonne dla zwykłego PC. Takie podejście można określić mianem *brute force*.

Program możemy przyspieszyć gdy analizę *brute force* poprzedzimy wstępną oceną, tj. dla każdego wolnego pola sprawdzamy ile numerków tam pasuje (nie powoduje prostego konfliktu). Wówczas okaże się, że liczba możliwych kombinacji jest dużo mniejsza np.: 2x4x5x2x7x3x2x7x5x2= 235 200. Przy takiej liczbie i porządnie napisanym programie da się przeanalizować wszystkie kombinacje w rozsądnym czasie.

Źródła

[Wiki-Sudoku] http://pl.wikipedia.org/wiki/Sudoku
[Wiki-Lista] http://pl.wikipedia.org/wiki/Lista