Instrukcja do zajęć laboratoryjnych

Język ANSI C (w systemie LINUX)

wersja: 2.01

Nr ćwiczenia:	8, 9	
Temat:	**Gra logiczna Master Mind**	
Cel ćwiczenia:	Celem ćwiczenia jest napisanie gry Master Mind umożliwiającej grę z komputerem.	
Wymagane przygotowanie teoretyczne:	Zasada gry w Master Mind-a, której opis zamieszczone poniżej.	
Sposób zaliczenia:	Sprawozdanie w formie pisemnej.	[**X**]
	Pozytywna ocena ćwiczenia przez prowadzącego pod koniec zajęć.	[]

1. Uwagi wstępne

Celem ćwiczenia jest napisanie prostej gry o nazwie *Master Mind*. Najbardziej popularna w Polsce była ona w drugiej połowie lata 70. oraz w początkach 80. Do dzisiaj jednak można w sklepach kupić jej kolejne wydania. Na poniższym zdjęciu pokazano jak ona wygląda w wersji „hardware-owej" (oczywiście forma plastyczna nie musi być taka sama u każdego producenta. Jednak zasady gry są zawsze takie same).

2. Zasady gry

Gra jest przeznaczona dla dwóch osób. Jeden z graczy staje się *kodującym*, drugi – *odgadującym*. Kodujący układa w `N` otworach dowolną kombinację kolorowych pionków. Ilość dostępnych kolorów wynosi `M` i zwykle powinno być `N < M`. Aby gra zbyt prosta ani zbyt monotonna zwykle `N` mieści się w przedziale 4-7 a `M` w przedziale 5-10. Kolory mogą się powtarzać. Poprawne są więc przykładowe kombinacje:

```
czerwony-żółty-zielony-czarny

czerwony-czerwony-zielony-zielony
```

Następnie odgadujący wprowadza swoją kombinację kolorów. W odpowiedzi kodujący analizuje wprowadzony kod i podaje informację o jego zgodności ze swoim kodem za pomocą białych i czarnych pionów (na zdjęciu powyżej są czerwone – jest to oczywiście kwestia umowy). Biały pion oznacza, iż kolor w kombinacji odgadującego zgadza się z kolorem w kombinacji kodującego, jednakże jest na złej pozycji. Czarny pion oznacza, iż w kombinacji odgadującego i kodującego na tej samej pozycji występuje ten sam kolor. Gra kończy się, gdy odgadujący odgadnie kombinację kodującego otrzymując `N` czarnych pionów (oznacza to, że wygrał) lub wykorzysta limit `K` ruchów i nie udało mu się odgadnąć kombinacji kodującego (oznacza to, że przegrał).

Aby lepiej zrozumieć zasady tej gry, przyjrzyjmy się przykładowej rozgrywce. Zakładamy, że `N=4`, `M=10` (cyfry od 0 do 9), `K=8`. W przykładzie poniżej nie posługujemy się kolorami tylko cyframi, jednak w żaden sposób nie zmienia to istoty gry. Ponadto zakładamy, że gramy z komputerem, którego zadaniem jest wylosowanie tajnego kodu oraz udzielanie grającemu odpowiedzi (podawanie ilości czarnych i białych pionów).

Numer ruchu	Kody	Piony czarne	Piony białe	Opis
-	**1 3 6 6**			Przykładowy kod podany przez komputer. Kod ten musi odgadnąć gracz.
1	1 1 1 3	●	○	Ten kod wprowadza gracz. W odpowiedzi komputer daje jeden pion czarny oraz jeden biały, ponieważ cyfra 1 zgadza się z pozycją cyfry 1 w kodzie wygenerowanym przez komputer, a cyfra 3 również występuje w tym kodzie jednak na innej pozycji. Oczywiście gracz nie wie o które cyfry chodzi - musi to odgadnąć na podstawie analizy swoich ruchów i otrzymanych na nie odpowiedzi.
2	1 4 3 4	●	○	Gracz założył, że cyfrą na właściwym miejscu jest 1 a na niewłaściwym 3. Dlatego też cyfrę 1 pozostawił na tej samej pozycji a cyfrę 3 przesunął na inną pozycję. W odpowiedzi komputer daje jeden pion czarny oraz jeden biały.
3	1 3 4 4	● ●		Gracz nadal zakłada, że na właściwym miejscu jest cyfra 1. Dodatkowo zakłada, że cyfrą na niewłaściwej pozycji jest 3 i wobec tego przesuwa ją na pozycję drugą. Odpowiedź otrzymana przez komputer (dwa czarne piony) pozwala mu ponadto z dużym prawdopodobieństwem stwierdzić, że w wylosowanej sekwencji nie ma cyfr 4.

	1 3 5 5	●●		Gracz, wiedząc już, że w wylosowanej sekwencji pierwsza i druga pozycją są już „pewne”, sprawdza kolejne cyfry. Okazuje się, że tym razem nie udało mu się ich poprawnie zgadnąć (gracz musiał w zasadzie zgadywać – nie ma bowiem żadnych przesłanek, aby wybrać te cyfry drogą logicznego rozumowania).
5	1 3 6 6	●●●●		Gracz tym razem podaje właściwe cyfry, otrzymuje cztery czarne piony i kończy zwycięsko grę.

Zwróćmy uwagę, że mimo bardzo dużej ilości możliwych do podania przez kodującego sekwencji (jak obliczyć ile ich jest dokładnie?) można w stosunkowo niewielu ruchach ją odgadnąć. Oczywiście aby tak się stało musimy kolejne swoje ruchy podawać tak, aby były zgodne z wszystkimi poprzednimi odpowiedziami podawanymi przez kodującego (zakładamy, że są one zawsze poprawne). Przykładowo w powyższym przykładzie w kroku 3 podanie sekwencji 8 8 9 9 jest całkowicie bezsensowne!

3. Opis zadania do wykonania

W ćwiczeniu chodzi o napisanie programu, który obsługuje jedynie zadania kodującego, czyli losuje układ liczb do odgadnięcia i generującej stosowne podpowiedzi (podaje ilości czarnych i białych pionów). Napisanie programu, który obsługiwałby również zadania odgadującego byłoby duuuuużo trudniejszym zadaniem. Czy potrafisz powiedzieć dlaczego?

Format komunikacji z programem zaproponowany w opisie poniżej jest tylko przykładowym pomysłem. Wykonujący ćwiczenie może zaproponować dowolny interfejs gry, umożliwiający jednak praktyczne (i wygodne) prowadzenie rozgrywki. Istotniejsze elementy, które powinny zostać zaimplementowane to:

- możliwość **prostego poprawiania** wprowadzonej kombinacji liczb przed ostatecznym zatwierdzeniem ruchu,
- nie usuwanie z ekranu **historii gry** (wcześniejszych ruchów grającego i podpowiedzi komputera). Zwróćmy uwagę, że gdybyśmy nie mieli dostępu do historii gry to byłoby nam bardzo trudno efektywnie grać i wybierać kolejne ruch nie na chybił trafił ale drogą logicznego rozumowania (chyba, że tą historię będziemy pamiętać, co nie jest oczywiście takie proste),
- możliwość ustalenia, jeszcze przed rozpoczęciem gry, **maksymalnej ilości kroków**, w których gra musi się zakończyć (sukcesem lub porażką),
- możliwość ustalenia, jeszcze przed rozpoczęciem gry, **ilości losowanych cyfr**,
- możliwość ustalenia, jeszcze przed rozpoczęciem gry, **zbioru cyfr**, które mogą być użyte do wygenerowania kodu do odgadnięcia (np. liczby z przedziału 2-7 lub 0-4 lub 0-9 itd.),
- możliwość uruchamiania aplikacji **w dwóch trybach** – w pierwszym wylosowana sekwencja cyfr pojawia się na ekranie (przydatne przy testowaniu działania programu), w drugim jest niewidoczna, powinna się jednak pojawić po zakończeniu gry,
- duży nacisk powinien zostać położony na **obsługę błędów**. Przykładowo, gdy użytkownik, wprowadzając kolejną sekwencję znaków, zamiast liczb będzie próbował podawać litery, program powinien ten fakt wykryć i zareagować odpowiednim komunikatem. Podobnie powinien zachować się, gdy zamiast wymaganej ilości znaków (liczb) podamy niewłaściwą ich ilość. Inne sytuacje „awaryjne” musisz sam przewidzieć i je oprogramować.

```
Czy wylosowany kod ma być widoczny, domyślnie N (T/N): T
Podaj poziom gry, domyślnie 5 (4-7):                    4
Podaj maksymalną ilość kroków, domyślnie 12:            8
Podaj ilość kolorów, domyślnie 8:                       5

Zaczynamy grę!

wylosowany  kod:  1 3 5 5    odpowiedź

krok 1:           1 1 2 3    (1,1)
krok 2:           4 3 3 5    (2,0)
krok 3:           6 5 5

Błąd: musisz podać cztery cyfry.

krok 3:           6 5 5 1    (1,2)
krok 4:           6135

Błąd: cyfry rozdzielaj białymi znakami.

krok 4:           6 1 a b

Błąd: dozwolone są tylko cyfry.

krok 4:           6 1 3 5    (1,2)
krok 5:           1 3 5 5    (4,0)

Wygrałeś w 5 ruchach!
```

4. Używanie tzw. sekwencji wyjścia

Przy okazji pisania programu zachęcamy do zapoznania się z możliwościami wyświetlania na terminalu tekstów w kolorach innych niż tylko czarny i biały. W kontekście gry MasterMind może to znacząco poprawić jej walory wizualne.

Aby zmusić terminal do wyświetlania kolorowych tekstów należy przekazać mu odpowiednie tzw. **sekwencje wyjścia** (ang. *escape sequences*). Każda taka sekwencja nakazuje terminalowi wykonanie określonych czynności (np. wyświetlenie tekstu w kolorze żółtym na niebieskim tle).

Sekwencja wyjścia zaczyna się od znaku ESC (stąd nazwa). Wszystkie kody sterujące zaczynają się od sekwencji `\033[` lub równoważnej `\e[`. Kody te interpretowane są przez polecenie `echo` oraz `printf` (`printf` daje zdecydowanie więcej możliwości. Funkcjonalnie odpowiada temu, który znamy ze standardowej biblioteki języka C). W przypadku tego pierwszego należy dodatkowo użyć przełącznika `-e`. Przykładowo wydanie następującej sekwencji wyświetla tekst kolorem żółtym na niebieskim tle. Dodatkowo tekst jest podkreślony:

```
$ printf '\033[4;33;44mżółte na niebieskim i podkreślone\n'
żółte na niebieskim i podkreślone
$
```

Zwróćmy uwagę, że po specyfikacji kolorów należy na końcu dodać literę `m`. Ponieważ kody sterujące przełączają kolory w sposób trwały, po wyświetleniu tekstu należy przywrócić kolory domyślne. Służy do tego kombinacja `\033[m`. Po poprawkach będzie więc:

```
$ printf '\033[4;33;44mżółte na niebieskim i podkreślone\033[m\n'
```

Jak łatwo się zorientować poszczególnym kolorom i stylom tekstu odpowiadają określone liczby. Pierwsza definiuje styl, dryga kolor czcionki a trzecia kolor tła. Mamy do dyspozycji osiem kolorów. Dla koloru czcionki są to: 30 (czarny), 31 (czerwony), 32 (zielony), 33 (żółty), 34

(niebieski), 35 (fioletowy), 36 (turkusowy), 37 (biały). Kolor tła określają liczby zaczynające się na 4. Zasada jest tu taka sama jak przy kolorach pierwszego planu. Pogrubienie oznaczane jest cyfrą 1 a podkreślenie cyfrą 4.

Aby zapoznać się z możliwościami jakie dają sekwencje wyjścia warto uruchomić poniższy skrypt powłoki:

```
#!/bin/sh
################################################################
# Nico Golde <nico(at)ngolde.de> Homepage: http://www.ngolde.de
# Last change: Mon Feb 16 16:24:41 CET 2004
################################################################

for attr in 0 1 4 5 7 ; do
    echo "-------------------------------------------------------------------"
    printf "ESC[%s;Foreground;Background - \n" $attr
    for fore in 30 31 32 33 34 35 36 37; do
        for back in 40 41 42 43 44 45 46 47; do
            printf '\033[%s;%s;%sm %02s;%02s  ' $attr $fore $back $fore $back
        done
    printf '\n'
    done
    printf '\033[0m'
done
```

Wśród kodów wyjścia są też takie, które pozwalają przemieszczać kursor w określone miejsce ekranu. Przykładowo wydanie poniższych poleceń czyści ekran i ustawia kursor w prawym górnym rogu ekranu.

```
$ printf ‘\033[2J’
$ printf ‘\033[0;0f’
```

Powyższe można też wykonać prościej:

```
$ printf \033c
```

Komenda ta przywraca domyślny stan terminala.

Z kolei komenda:

```
$ printf '\033[5;20H'




                    root@mykonos:~#
```

Pozwala przemieścić kursor do linii 5 i do kolumny 20. Wykorzystując te możliwości łatwo jest samodzielnie napisać funkcję typu `gotoxy(x,y)`. Inne przydatne kody to:

```
\033[xG - przesunięcie kursora do kolumny X
\033[nA - przesunięcie kursora o n znaków w górę
\033[nB - przesunięcie kursora o n znaków w dół
\033[nC - przesunięcie kursora o n znaków w prawo
\033[nD - przesunięcie kursora o n znaków w lewo
```

Więcej szczegółów na temat kodów sterujących można znaleźć w dokumentacji systemowej `man console_codes(4)`.

Sekwencje wyjścia można oczywiście używać w programach w języku C. Zasada jest tu analogiczna jak dla skryptów powłoki. Patrz poniżej:

```
int main(int argc, char *argv[]) {
  clrscr();
  return 0;
```

```
}

void clrscr(void) {
  printf("\033[2J");    /* Czyści bieżący ekran                       */
  printf("\033[0;0f");  /* Ustawia kursor w lewym górnym rogu ekranu */
}
```

5. Sprawozdanie

Sprawozdanie powinno zawierać następujące elementy:

- wydrukowany kod źródłowy programu (do wydruku użyć koniecznie takiej czcionki: `Courier New 8pt`). Plik źródłowy musi być **rozsądnie skomentowany** oraz **czytelnie sformatowany** (możesz użyć jakiegoś programu narzędziowego, który pomoże ładnie sformatować tekst źródłowy). Nieczytelny kod nie będzie sprawdzany !
- plik *makefile* do kompilacji programu,
- dyskusję rozwiązania i rzeczowe uwagi autora programu.