Programovanie

Návody na cvičenia (študent)

Miroslav Biňas, Marek Paralič, Ema Pietriková

Obsah

Časť I

Scenáre cvičení

1 First Steps with Karel the Robot

1.1 Ciele

1. Osvojiť si správanie sa robota Karla pomocou základných príkazov.
2. Naučiť sa riadiť robota Karla pomocou jednoduchých programov.
3. Naučiť sa vytvárať nové funkcie v jazyku C.
4. Naučiť sa preložiť a spustiť vytvorený program v prostredí operačného systému Linux.

1.2 Úvod

Na tomto cvičení spoznáte svet robota Karla a naučíte sa ho ovládať príkazmi, ktorým Karel rozumie od svojho výrobcu. Pomocou týchto príkazov budete vytvárať jednoduché programy a pomáhať Karlovi riešiť jednoduché úlohy. Okrem toho sa naučíte rozširovať slovník príkazov robota Karla o nové príkazy (vlastné funkcie), aby ste mohli písať svoje programy efektívnejšie.

1.3 Postup

Krok č. 1

V prvom kroku sa zoznámite s robotom Karlom a svetom, v ktorom sa nachádza. Vytvoríte pre Karla jednoduchý program, pomocou ktorého si osvojíte jeho základné funkcie. Následne sa naučíte, ako výsledný program preložiť a spustiť.

Úloha 1.1

Prihláste sa na server omega.tuke.sk so svojimi prihlasovacími údajmi.

Úloha 1.2

Vo svojom domovskom priečinku si vytvorte priečinok s názvom Karel, do ktorého budete ukladať svoje kódy programov pre robota Karla.

Úloha 1.3

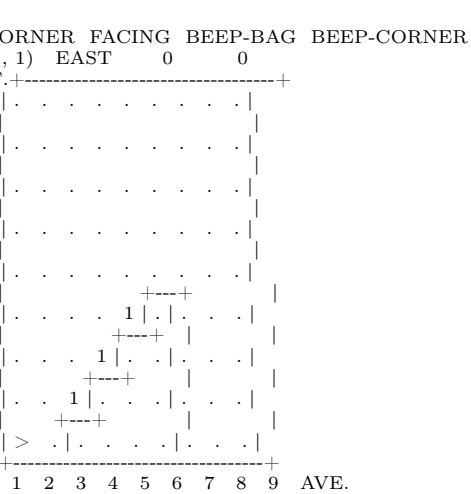
Vytvorte súbor stairs.kw, do ktorého Karla umiestníte, a v ktorom bude problém riešiť.

Poznámka	Obsah súboru sveta nájdete v prílohe v zozname svetov.

Úloha 1.4

Vytvorte pre Karla program, pomocou ktorého mu pomôžete vyšplhať sa po schodoch nahor, až na najvyšší stupienok. Cestou pomôžte Karlovi pozbierať všetky značky, ktoré na schodoch nájde. Po vyšplhaní sa na najvyšší schod Karel všetky značky položí. Pre riešenie tejto úlohy použite vami vytvorený svet stairs.kw.

Počiatočná situácia



Koncová situácia

```
 31
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (6, 5)  EAST      0         3
ST.+-----------------------------------+
 9 |.  .  .  .  .  .  .  .  .  .|
   |                             |
 8 |.  .  .  .  .  .  .  .  .  .|
   |                             |
 7 |.  .  .  .  .  .  .  .  .  .|
   |                             |
 6 |.  .  .  .  .  .  .  .  .  .|
   |                             |
 5 |.  .  .  .  .  >  .  .  .  .|
   |              +---+          |
 4 |.  .  .  .  .|.  |.  .  .  .|
   |           +---+   |         |
 3 |.  .  .  .|.  .|.  .  .  .|
   |        +---+      |         |
 2 |.  .  .|.  .  .|.  .  .  .|
   |     +---+         |         |
 1 |.  .|.  .  .  .|.  .  .  .|
   +-----------------------------------+
     1  2  3  4  5  6  7  8  9  AVE.
```

Poznámka Program napíšte v niektorom z dostupných textových editorov (napr. Vim, mcedit, nano alebo joe). Názov súboru, do ktorého napíšete váš program, odvoďte od názvu súboru s mapou zmenou prípony na .c (napr. ak sa mapa nachádza v súbore stairs.kw, tak program uložte do súboru s názvom stairs.c). Do programu vložte tieto riadky, ktoré budú predstavovať jeho kostru:

```
1 #include <karel.h>
2
3     int main(){
4
5     return 0;
6 }
```

Váš program sa bude nachádzať medzi riadkami 3 a 5.

Pri riešení tejto úlohy využite tieto Karlove príkazy:

- movek() - Presunie Karla o jednu pozíciu vpred vzhľadom na smer, ktorým je Karel otočený. Pokiaľ sa Karel pokúsi pohnúť smerom, ktorým nemôže (v ceste stojí prekážka alebo sa nachádza na okraji sveta), program sa zastaví.
- pickBeeper() - Pokiaľ Karel stojí na pozícii, na ktorej sa nachádza značka, Karel ho zodvihne a vloží do svojho batohu.
- putBeeper() - Pokiaľ má Karel vo svojom batohu značku, položí ju na pozíciu, na ktorej sa aktuálne nachádza.
- turnLeft() - Po zadaní tohto príkazu sa Karel otočí na mieste o 90 stupňov vľavo.

- turnOn() - Príkaz, ktorý Karla zapne. Príkaz má jeden povinný parameter, ktorým je umiestnenie sveta, v ktorom Karel bude riešiť svoju úlohu. Príklad zapnutia Karla a jeho umiestnenia do sveta nachádzajúceho sa v súbore stairs.kw môže teda vyzerať nasledovne:

```
1 turnOn( "stairs.kw" );
```

- turnOff() - Po vykonaní všetkých úloh tento príkaz zabezpečí korektné ukončenie Karlovej úlohy.

Poznámka V programovacom jazyku C záleží na veľkosti písmen použitých v názvoch funkcií.

Poznámka Korektné ukončenie Karla zabezpečuje funkcia turnOff(). Ak sa Karel neukončí korektne, zadajte príkaz reset.

Úloha 1.5

Preložte vytvorený program pomocou prekladača compile. V prípade, že prekladač objaví chyby, opravte ich.

Program preložíte nasledovne:

```
1 $ compile stairs.c
```

pričom súbor stairs.c predstavuje vami vytvorený program pre Karla.

Poznámka Znak '$' nie je súčasťou príkazu, ale symbolizuje len použitie príkazového riadku. Pri zadávaní (alebo kopírovaní) uvedeného príkazu ho preto vynechajte.

Úloha 1.6

Spustite preložený program.

Pokiaľ preklad prebehol bez chýb, v priečinku, kde ste program preložili, vznikol súbor s názvom vášho zdrojového programu, ale s príponou .run. Tento program spustíte nasledovne:

```
1 $ ./nazov.run
```

Úloha 1.7

Overte, ako sa Karel správa v prípadoch, keď chce prejsť cez stenu, alebo keď sa pokúsi položiť značku a v batohu už žiadnu nemá, alebo keď sa pokúsi vziať značku z rohu, na ktorom sa žiadna značka nenachádza.

Krok č. 2

V druhom kroku rozšírite slovník príkazov robota Karla o vlastné príkazy (funkcie). Upravíte riešenie príkladu z kroku 1 a zjednodušíte ho vytvorením dvoch nových príkazov.

Úloha 2.1

Vytvorte funkciu turnRight(), ktorá zabezpečí otočenie robota Karla o 90 stupňov vpravo.

Poznámka Pomocou volania funkcie setStepDelay() môžete nastaviť časovú dĺžku vykonávania jedného kroku robota Karla. Pokiaľ ju nastavíte na hodnotu 0, dôjde k okamžitému vykonaniu tohto kroku. Zaujímavý efekt môžete dosiahnuť vo funkcii turnRight(), kde na jej začiatku nastavíte dĺžku vykonávania kroku robota Karla na hodnotu 0 a na jej konci ju vrátite na pôvodnú hodnotu. Vykonanie funkcie turnRight() sa bude javiť tak, akoby sa vykonala okamžite.

Úloha 2.2

Vytvorte funkciu climbStair(), pomocou ktorej Karel vystúpi o jeden schod vyššie.

Úloha 2.3

Upravte hlavnú funkciu programu main() tak, aby ste v nej využili vytvorené funkcie turnRight() a climbStair().

Úloha 2.4

Overte správnosť upraveného riešenia jeho preložením a spustením.

Úloha 2.5

Porovnajte počet riadkov výsledného programu pred použitím funkcií a po ich použití.

1.4 Doplňujúce úlohy

1. Robot Karel sa chce zúčastniť olympiády pre robotov (tzv. robolympiády). Jedna z disciplín, ktorej sa je možné v rámci olympiády zúčastniť, je prekážkový beh. Napíšte pre Karla program, pomocou ktorého zvládne zdolať rozcvičkovú prekážkovú trať nachádzajúcu sa v súbore sveta training.kw. Pre úspešné zvládnutie prekážkového behu vytvorte funkciu jumpOver(), pomocou ktorej Karel preskočí práve jednu prekážku. Na jej vytvorenie využite

ľubovoľné už vytvorené funkcie z riešení predchádzajúcich úloh. Všetky prekážky sú rovnako vysoké a Karel potrebuje trať prebehnúť z počiatočnej pozície [1,1] do koncovej pozície [5,1].

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)   EAST       0         0
ST.+-------------------+
 2 | .   .   .   .   . |
   |   |   |   |   |   |
 1 | > | . | . | . | . |
   +-------------------+
     1   2   3   4   5   AVE.
```

Koncová situácia

```
 44
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (5, 1)   EAST       0         0
ST.+-------------------+
 2 | .   .   .   .   . |
   |   |   |   |   |   |
 1 | . | . | . | . | > |
   +-------------------+
     1   2   3   4   5   AVE.
```

Poznámka	Obsah súboru sveta nájdete v prílohe v zozname svetov.

2. Cesta vedúca do areálu Technickej univerzity je za tie roky značne poškodená a vyznačuje sa väčším (občas aj menším) počtom dier v nej. Naprogramujte preto robota Karla tak, aby každú dieru, ktorú v ceste nájde, vyplnil značkou. Cesta, ktorú má Karel vyplniť, sa nachádza v súbore road.kw. Karel sa na začiatku nachádza na počiatočnej pozícii [1,2] a po zaplnení všetkých dier sa bude nachádzať na pozícii [5,2].

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 2)   EAST       2         0
ST.+-------------------+
 2 | >   .   .   .   . |
   |---+   +---+   +---|
 1 | . | . | . | . | . |
   +-------------------+
     1   2   3   4   5   AVE.
```

Poznámka	Obsah súboru sveta nájdete v prílohe v zozname svetov.

Koncová situácia

```
 26
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (5, 2)  EAST       0         0
ST.+--------------------+
 2 | .   .   .   .   > |
   |---+   +---+   +---|
 1 | . | 1 | . | 1 | . |
   +-------------------+
     1   2   3   4   5   AVE.
```

1.5 Ďalšie zdroje

- Robot Karel (http://sk.wikipedia.org/wiki/Karel_(programovací_jazyk)) - stránka na wikipédii (http://www.wikipedia.org) venovaná robotovi Karlovi
- Konfiguračný súbor pre editor *Vim* - .vimrc
- Rudolf Pecinovský: Základy algoritmizace (http://publikace.pecinovsky.cz/Zaklady_algoritmizace.pdf) - kapitola 6.1 a 11 (celá kapitola)
- Pavel Herout: Učebnice jazyka C (1. díl) (http://www.martinus.sk/?uItem=74741) - kapitoly 9.2 (úvod), 9.2.1, 9.2.2, 9.2.4, 9.2.5, kapitoly 5.1 a 5.4

2 Karel and his Sensors

2.1 Ciele

1. Naučiť sa vetviť tok programu na základe podmienok.
2. Naučiť sa používať v programoch logické cykly.
3. Osvojiť si príkazy break a continue pre riadenie behu cyklov.

2.2 Úvod

Na tomto cvičení sa zoznamíte so senzormi, ktorými vybavil robota Karla jeho výrobca. Na základe ich hodnôt naučíte Karla rozhodovať sa a vetviť jeho program.

2.3 Postup

Krok č. 1 – Karel opravuje cestu

V prvom kroku pomôžete robotovi Karlovi opraviť cestu vedúcu do areálu Technickej univerzity, ktorá je za tie roky značne poškodená a vyznačuje sa väčším (občas aj menším) počtom dier v nej. Naprogramujete preto robota Karla tak, aby každú dieru, ktorú v ceste nájde, vyplnil značkou. Karel sa na začiatku bude nachádzať na pozícii [1,2] a po zaplnení všetkých dier sa bude nachádzať na konci cesty daného sveta.

Úloha 1.1

Otvorte si riešenie doplňujúcej úlohy č. 2 z minulého cvičenia alebo ho nájdite v prílohe riešení ako súbor road.c. Pôvodná cesta, ktorú mal Karel vyplniť, sa nachádza v súbore sveta road.kw, ktorý by ste už z minulého cvičenia mali mať vytvorený.

Úloha 1.2

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 2)  EAST       2         0
ST.+--------------------+
 2 | >   .   .   .   . |
   |---+   +---+   +---|
 1 | . | . | . | . | . |
   +--------------------+
     1   2   3   4   5   AVE.
```

Koncová situácia

```
 26
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (5, 2)  EAST       0         0
ST.+--------------------+
 2 | .   .   .   .   > |
   |---+   +---+   +---|
 1 | . | 1 | . | 1 | . |
   +--------------------+
     1   2   3   4   5   AVE.
```

Upravte program tak, aby Karel vedel vyplniť vždy všetky diery nezávisle na ich množstve až do konca cesty.

Pre úspešné vyriešenie tejto úlohy môžete využiť tieto Karlove senzory:

- frontIsClear() - Vracia hodnotu 1, ak sa pred Karlom nenachádza stena. V opačnom prípade vracia hodnotu 0.
- frontIsBlocked() - Vracia hodnotu 1, ak sa pred Karlom nachádza stena. V opačnom prípade vracia hodnotu 0.
- rightIsClear() - Vracia hodnotu 1, ak sa vpravo od Karla nenachádza stena. V opačnom prípade vracia hodnotu 0.
- rightIsBlocked() - Vracia hodnotu 1, ak sa vpravo od Karla nachádza stena. V opačnom prípade vracia hodnotu 0.

Svoje riešenie tejto úlohy môžete otestovať na svetoch road2.kw a road3.kw, ktoré nájdete v prílohe v zozname svetov.

Úloha 1.3

Upravte program tak, aby Karel počas plátania dier na ceste vedúcej do areálu Technickej univerzity nevypĺňal aj tie diery, ktoré už sú vyplnené.

Pre úspešné vyriešenie tejto úlohy môžete využiť tieto Karlove senzory:

- beepersPresent() - Vracia hodnotu 1, ak Karel stojí na rohu, na ktorom sa nachádza značka. V opačnom prípade vracia hodnotu 0.
- noBeepersPresent() - Vracia hodnotu 1, ak Karel stojí na rohu, na ktorom sa nenachádza značka. V opačnom prípade vracia hodnotu 0.

Svoje riešenie tejto úlohy môžete otestovať na svetoch road4.kw a road5.kw, ktoré nájdete v prílohe v zozname svetov.

Krok č. 2 – Robolympiáda

V tomto kroku pomôžete Karlovi zúčastniť sa a uspieť v olympiáde pre robotov. Jednou z disciplín, ktorej sa je možné v rámci olympiády zúčastniť, je prekážkový beh. Napíšete preto pre Karla program, pomocou ktorého zvládne zdolať akúkoľvek prekážkovú trať, ktorú orgranizátori pre účastníkov olympiády pripravia. Karel bude vždy štartovať z pozície [1,1].

Úloha 2.1

Otvorte si riešenie doplňujúcej úlohy č. 1 z minulého cvičenia alebo ho nájdite v prílohe riešení ako súbor olympics.c. Tréningová prekážková trať sa nachádza v súbore sveta training.kw, ktorý by ste už z minulého cvičenia mali mať vytvorený.

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)   EAST       0         0
ST.+-------------------+
 2 | .   .   .   .   . |
   |   |   |   |   |   |
 1 | > | . | . | . | . |
   +-------------------+
     1   2   3   4   5   AVE.
```

Koncová situácia

```
 44
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (5, 1)   EAST       0         0
ST.+-------------------+
 2 | .   .   .   .   . |
   |   |   |   |   |   |
 1 | . | . | . | . | > |
   +-------------------+
     1   2   3   4   5   AVE.
```

Úloha 2.2

Upravte program tak, aby sa Karel zastavil v behu vtedy, keď nájde značku, pričom značka sa v Karlovej ceste bude vždy nachádzať.

Svoje riešenie tejto úlohy si môžete otestovať na prekážkovej dráhe olympics1.kw, ktorú nájdete v prílohe v zozname svetov.

Úloha 2.3

Upravte program tak, aby Karel zvládol prebehnúť ľubovoľný počet prekážok, ktoré môžu byť od seba ľubovoľne vzdialené. Karel svoj beh zastaví vtedy, keď

vo svojej trati natrafí na značku, pričom značka sa bude v Karlovej ceste vždy nachádzať.

Svoje riešenie tejto úlohy si môžete otestovať na prekážkovej dráhe olympics2.kw, ktorú nájdete v prílohe v zozname svetov.

2.4 Doplňujúce úlohy

1. Vytvorte pre Karla program, pomocou ktorého premení aktuálny prázdny svet na šachovnicu. Svet má vždy párny počet políčok. Pre otestovanie tejto úlohy použite súbory svetov empty1.kw a empty2.kw.

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)   EAST     99        0
ST.+----------------------+
 4 | .  .  .  .  .  .  . |
   |                      |
 3 | .  .  .  .  .  .  . |
   |                      |
 2 | .  .  .  .  .  .  . |
   |                      |
 1 | >  .  .  .  .  .  . |
   +----------------------+
     1  2  3  4  5  6  AVE.
```

Koncová situácia

```
 48
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 6)   NORTH    87        1
ST.+----------------------+
 4 | ^  .  1  .  1  .  . |
   |                      |
 3 | .  1  .  1  .  1  . |
   |                      |
 2 | 1  .  1  .  1  .  . |
   |                      |
 1 | .  1  .  1  .  1  . |
   +----------------------+
     1  2  3  4  5  6  AVE.
```

2. Vytvorte program pre robota Karla, pomocou ktorého Karel zozbiera všetky značky, ktoré sa v ňom nachádzajú. Karel ich všetky umiestni v juhovýchodnom rohu miestnosti a sám sa nakoniec presunie na juhozápadnú pozíciu. Pre otestovanie tejto úlohy použite súbory svetov collector1.kw a collector2.kw.

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)  EAST       0         0
ST.+--------------------+
 5 |.   2   .   .   . |
   |                   |
 4 |.   .   .   .   . |
   |                   |
 3 |.   .   .   .   2 |
   |                   |
 2 |.   .   1   .   . |
   |                   |
 1 |>   .   .   3   . |
   +-------------------+
     1   2   3   4   5   AVE.
```

Koncová situácia

```
 70
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)  WEST       0         0
ST.+--------------------+
 5 |.   .   .   .   . |
   |                   |
 4 |.   .   .   .   . |
   |                   |
 3 |.   .   .   .   . |
   |                   |
 2 |.   .   .   .   . |
   |                   |
 1 |<   .   .   .   8 |
   +-------------------+
     1   2   3   4   5   AVE.
```

2.5 Ďalšie zdroje

- Rudolf Pecinovský: Základy algoritmizace (http://publikace.pecinovsky.cz/Zaklady_algoritmizace.pdf) - kapitola 10 (úvod), 10.2, 10.3, 11 (celá kapitola)
- Pavel Herout: Učebnice jazyka C (1. díl) (http://www.martinus.sk/?uItem=74741) - kapitoly 5.1, 5.4 a 5.5

3 Karel Solves Daily Problems

3.1 Ciele

1. Precvičiť si vetvenie toku programu na základe podmienok v programovacom jazyku C.
2. Naučiť sa používať logické cykly v programovacom jazyku C.
3. Osvojiť si príkazy break a continue pre riadenie behu cyklov.

3.2 Úvod

Na tomto cvičení si opäť precvičíte rozhodovanie v Karlových programoch na základe senzorov, ktoré obsahuje. Úlohou tohto cvičenia je naučiť sa vytvárať jednoduché cykly a s ich pomocou efektívne vyriešiť problém.

3.3 Postup

Krok č. 1

V prvom kroku vytvoríte niekoľko jednoduchých funkcií, ktoré môžete neskôr využiť v niektorých úlohách.

Úloha 1.1

Vytvorte funkciu turnRight(), po vykonaní ktorej bude robot Karel otočený vpravo vzhľadom na jeho východziu pozíciu. Pri tvorbe funkcie použite niektorý z cyklov.

Úloha 1.2

Vytvorte funkciu turnToNorth(), po vykonaní ktorej bude robot Karel otočený na sever.

Úloha 1.3

Vytvorte funkciu pickAllBeepers(), po vykonaní ktorej robot Karel vezme všetky značky z aktuálnej pozície a vloží si ich do batohu.

Úloha 1.4

Vytvorte funkciu plantBeepers(). Po jej zavolaní začne Karel "sadiť"značky postupne pred seba krok za krokom, pričom jednu značku zasadí v každom kroku. Ak cestou vpred Karel nájde na pozícii inú značku alebo sa dopredu už nebude môcť pohnúť, funkcia sa ukončí.

Úloha 1.5

Vytvorte funkciu findExitOnLeft(). Po jej zavolaní sa Karel rozbehne vpred a zastaví sa vtedy, keď sa už nebude môcť pohnúť vpred. V prípade, ak sa po Karlovej ľavej ruke nachádza voľná pozícia, otočí sa vľavo a zastaví sa.

Krok č. 2

V tomto kroku si zopakujete prácu s robotom Karlom a svetom, v ktorom sa nachádza, s využitím vlastných funkcií. Následne sa naučíte, ako program v jazyku C preložiť a spustiť v prostredí operačného systému Linux.

Úloha 2.1

Vytvorte program marker.c, pomocou ktorého Karel označkuje všetky štyri rohy sveta. Karel sa na začiatku nachádza na pozícii [1,1] a je nasmerovaný na východ. Po označení všetkých rohov sa vráti na svoju východziu pozíciu. Vaše riešenie otestujte svete empty.kw, ktorý nájdete v prílohe v zozname svetov.

Počiatočná situácia

```
0
CORNER  FACING  BEEP-BAG  BEEP-CORNER
(1, 1)   EAST      4         0
ST.+----------------------+
 6 | .   .   .   .   .   . |
   |                       |
 5 | .   .   .   .   .   . |
   |                       |
 4 | .   .   .   .   .   . |
   |                       |
 3 | .   .   .   .   .   . |
   |                       |
 2 | .   .   .   .   .   . |
   |                       |
 1 | >   .   .   .   .   . |
   +----------------------+
     1   2   3   4   5   6   AVE.
```

Úloha 2.2

Preložte vytvorený program pomocou prekladača gcc. V prípade, že prekladač objaví chyby, opravte ich.

Koncová situácia

```
27
CORNER  FACING  BEEP-BAG  BEEP-CORNER
(1, 1)  SOUTH      0         1
ST.+-----------------------+
 6 | 1   .   .   .   .   1 |
   |                       |
 5 | .   .   .   .   .   . |
   |                       |
 4 | .   .   .   .   .   . |
   |                       |
 3 | .   .   .   .   .   . |
   |                       |
 2 | .   .   .   .   .   . |
   |                       |
 1 | v   .   .   .   .   1 |
   +-----------------------+
     1   2   3   4   5   6   AVE.
```

Program preložíte nasledovne:

```
1 $ gcc marker.c -o karel -lkarel -lcurses
```

pričom:

- súbor marker.c predstavuje vami vytvorený program pre Karla,
- prepínač -o zabezpečí, aby sa výsledný súbor, ktorý bude reprezentovať spustiteľný program, volal *karel*,
- voľba -lkarel zabezpečí, aby bola pri preklade použitá knižnica pre prácu s robotom Karlom, a
- voľba -lcurses zabezpečí, aby bola pri preklade použitá aj knižnica *ncurses* pre vykresľovanie sveta, v ktorom sa Karel bude nachádzať.

Poznámka Pokiaľ by ste pri preklade nepoužili prepínač -o, výsledný spustiteľný program by sa nachádzal v súbore a.out.

Poznámka Znak '$' nie je súčasťou príkazu, ale symbolizuje len použitie príkazového riadku. Pri zadávaní (alebo kopírovaní) uvedeného príkazu ho preto vynechajte.

Poznámka Ak chcete programovať na vlastnom počítači s nainštalovaným operačným systémom Linux, potrebujete mať nainštalovaný balíček libncurses5-dev.

Úloha 2.3

Spustite preložený program.

Pokiaľ ste spustili prekladač s parametrom -o karel, program spustíte nasledovne:

```
$ ./karel
```

Úloha 2.4

V programe marker.c vytvorte funkciu runMile(), po zavolaní ktorej sa Karel prejde o 5 krokov vpred a upravte hlavnú funkciu programu main() tak, aby ste v nej využili volanie vytvorenej funkcie runMile().

Poznámka V programovacom jazyku *C* záleží na veľkosti písmen použitých v názvoch funkcií.

Poznámka Nezabudnite, že pri volaní jednej funkcie z druhej musí byť funkcia na mieste svojho volania už známa.

Úloha 2.5

Upravte program tak, aby Karel počas značkovania rohov sveta neznačkoval aj tie rohy, ktoré už sú označené.

Svoje riešenie tejto úlohy môžete otestovať na svete empty1.kw.

Krok č. 3

Karlove prípravy na RobOlympiádu vrcholia. V tomto kroku pomôžete Karlovi dotiahnuť prípravy a naučiť ho zdolať v rámci prekážkového behu prekážky ľubovoľne od seba vzdialené aj ľubovoľne vysoké prekážky. Na začiatku každého behu sa Karel vždy bude nachádzať na pozícii [1,1]. Karlov beh sa opäť zastaví vtedy, keď vo svojom behu nájde značku.

Úloha 3.1

Otvorte si riešenie olympics z minulého cvičenia.

Úloha 3.2

Upravte program tak, aby Karel zvládol prebehnúť ľubovoľne vysoké prekážky. Karel svoj beh zastaví vtedy, keď vo svojej trati natrafí na značku, pričom značka sa bude v Karlovej ceste vždy nachádzať.

Svoje riešenie tejto úlohy si môžete otestovať na nasledujúcich mapách, ktoré predstavujú predkolá veľkého finále RobOlympiády: olympics3.kw, olympics4.kw.

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)  EAST       0         0
ST.+---------------------------------------+
 6 | .   .   .   .   .   .   .   .   .   . |
   |                                       |
 5 | .   .   .   .   .   .   .   .   .   . |
   |                                       |
 4 | .   .   .   .   .   .   .   .   .   . |
   |                                       |
 3 | .   .   .   .   .   .   .   .   .   . |
   |                                       |
 2 | .   .   .   .   .   .   .   .   .   . |
   |                                       |
 1 | >   .   .   .   .   .   .   .   .   . |
   +---------------------------------------+
     1   2   3   4   5   6   7   8   9   10  AVE.
```

Koncová situácia

```
 91
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (10, 1)  EAST      0         1
ST.+---------------------------------------+
 6 | .   .   .   .   .   .   .   .   .   . |
   |                                       |
 5 | .   .   .   .   .   .   .   .   .   . |
   |                                       |
 4 | .   .   .   .   .   .   .   .   .   . |
   |                                       |
 3 | .   .   .   .   .   .   .   .   .   . |
   |                                       |
 2 | .   .   .   .   .   .   .   .   .   . |
   |                                       |
 1 | .   .   .   .   .   .   .   .   .   > |
   +---------------------------------------+
     1   2   3   4   5   6   7   8   9   10  AVE.
```

Úloha 3.3

Upravte program tak, aby robot Karel vedel preskočiť aj ľubovoľne široké prekážky. Karel svoj beh zastaví vtedy, keď vo svojej trati natrafí na značku, pričom značka sa bude v Karlovej ceste vždy nachádzať.

Svoje riešenie tejto úlohy si môžete otestovať na nasledujúcich mapách, ktoré predstavujú posledné kolá veľkého finále RobOlympiády: olympics5.kw, olympics6.kw.

3.4 Doplňujúce úlohy

1. Upravte predchádzajúce programy tak, aby robot Karel smeroval z východu na západ.

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)   EAST       0         0
ST.+-----------------------------------------+
 6 | .   .   .   .   .   .   .   .   .   . |
   |           |                 |
 5 | .   .   .   . | .   .   .   .   .   . |
   |           |       +-------+         |
 4 | .   .   .   . | .   . | .   . | .   . |
   |           |       |       |   |   |
 3 | .   .   .   . | .   . | .   . | . | . |
   |   +---+       |       |       |   |   |
 2 | . | . | .   . | .   . | .   . | . | . |
   |   |   |       |       |       |   |   |
 1 | > | . | .   . | .   . | .   . | 1 | . |
   +-----------------------------------------+
     1   2   3   4   5   6   7   8   9   10  AVE.
```

Koncová situácia

```
 54
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (9, 1)   EAST       0         1
ST.+-----------------------------------------+
 6 | .   .   .   .   .   .   .   .   .   . |
   |           |                 |
 5 | .   .   .   . | .   .   .   .   .   . |
   |           |       +-------+         |
 4 | .   .   .   . | .   . | .   . | .   . |
   |           |       |       |   |   |
 3 | .   .   .   . | .   . | .   . | . | . |
   |   +---+       |       |       |   |   |
 2 | . | . | .   . | .   . | .   . | . | . |
   |   |   |       |       |       |   |   |
 1 | . | . | .   . | .   . | .   . | > | . |
   +-----------------------------------------+
     1   2   3   4   5   6   7   8   9   10  AVE.
```

Poznámka Pre otestovanie týchto programov je potrebné modifikovať súbory svetov.

2. Vytvorte program pre robota Karla, pomocou ktorého Karel prejde bludiskom. Koniec bludiska je označený značkou, pričom v bludisku je vždy iba jedna značka. Riešenie tejto úlohy si môžete overiť na mapách maze1.kw, maze2.kw a maze3.kw.
3. Robot Karel sa vydal hľadať poklad, pričom potrebuje sledovať špirálu smerom dovnútra, až kým nenájde poklad. Keď poklad nájde (značku), dá si ho do batohu. Karel vždy začína na pozícii [1,1] a nevie, aké dlhé sú steny. Riešenie tejto úlohy si môžete overiť na mape treasurehunt.kw.

Poznámka Použite iné riešenie ako v predošlej úlohe (bludisko).

4. Karel bol najatý, aby pomohol opraviť škody spôsobené za tie roky na Košickom hrade. Karel má teda opraviť sadu kamenných pilierov (samozrejme

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)  EAST       0         0
ST.+----------------------+
 6 | .   .   .   .   . | 1 |
   |----   |   |   +---+   |
 5 | .   . | . | . | .   . |
   |   +---+   |   |   +---|
 4 | . | .   . | . | . | . |
   |   |   ----+---+   |   |
 3 | . | .   .   . | .   . |
   |   +---+----   +---+   |
 2 | .   . | .   .   . | . |
   |   +---+   ----+   |   |
 1 | > | .   .   . | .   . |
   +----------------------+
     1   2   3   4   5   6   AVE.
```

Koncová situácia

```
 181
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (6, 6)  NORTH      0         1
ST.+----------------------+
 6 | .   .   .   .   . | ^ |
   |----   |   |   +---+   |
 5 | .   . | . | . | .   . |
   |   +---+   |   |   +---|
 4 | . | .   . | . | . | . |
   |   |   ----+---+   |   |
 3 | . | .   .   . | .   . |
   |   +---+----   +---+   |
 2 | .   . | .   .   . | . |
   |   +---+   ----+   |   |
 1 | . | .   .   . | .   . |
   +----------------------+
     1   2   3   4   5   6   AVE.
```

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)  EAST       0         0
ST.+------------------------------+
 8 | .   .   .   .   .   .   .   . |
   |   +----------------------+   |
 7 | . | .   .   .   .   .   . | . |
   |   |   +--------------+   |   |
 6 | . | . | .   .   .   . | . | . |
   |   |   |   +--------+   |   |   |
 5 | . | . | . | 1   . | . | . | . |
   |   |   |   +----    |   |   |   |
 4 | . | . | .   .   . | . | . | . |
   |   |   +------------+   |   |   |
 3 | . | . |   .   .   . | . | . |
   |   +----------------------+   |   |
 2 | .   .   .   .   .   .   . | . |
   |------------------------------+   |
 1 | >   .   .   .   .   .   .   . |
   +------------------------------+
     1   2   3   4   5   6   7   8   AVE.
```

reprezentovaných značkami), v ktorých niektoré kamene chýbajú. Po skon-

Koncová situácia

```
78
CORNER  FACING  BEEP-BAG  BEEP-CORNER
(4, 5)  WEST      1         0
ST.+--------------------------------+
 8 | .   .   .   .   .   .   .   . |
   |   +-----------------------+   |
 7 | . | .   .   .   .   .   . | . |
   |   |   +---------------+   |   |
 6 | . | . | .   .   .   . | . | . |
   |   |   |   +-------+   |   |   |
 5 | . | . | . | <   . | . | . | . |
   |   |   |   +----   |   |   |   |
 4 | . | . | .   .   . | . | . | . |
   |   |   +-----------+   |   |   |
 3 | . | .   .   .   .   . | . | . |
   |   +-------------------+   |   |
 2 | .   .   .   .   .   .   . | . |
   |---------------------------+   |
 1 | .   .   .   .   .   .   .   . |
   +-------------------------------+
     1   2   3   4   5   6   7   8   AVE.
```

čení opráv budú všetky chýbajúce diery v stĺpoch opäť zaplnené. Riešenie tejto úlohy si môžete overiť na mapách stonemason1.kw a stonemason2.kw.

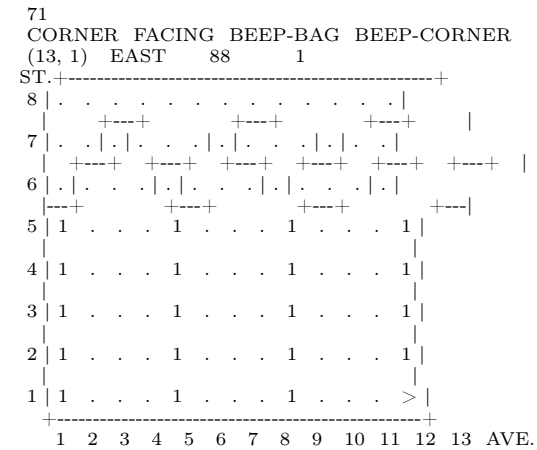
Počiatočná situácia

```
0
CORNER  FACING  BEEP-BAG  BEEP-CORNER
(1, 1)  EAST      99        0
ST.+-----------------------------------------------+
 8 | .   .   .   .   .   .   .   .   .   .   .   . |
   |       +---+           +---+           +---+       |
 7 | .   . | . | .   .   . | . | .   .   . | . | .   . |
   |   +---+   +---+   +---+   +---+   +---+   +---+   |
 6 | . | .   .   . | . | .   .   . | . | .   .   . | . |
   |---+           +---+           +---+           +---|
 5 | 1   .   .   .   .   .   .   1   .   .   .   1 |
   |                                               |
 4 | .   .   .   .   1   .   .   .   .   .   .   . |
   |                                               |
 3 | .   .   .   .   .   .   .   .   1   .   .   1 |
   |                                               |
 2 | .   .   .   .   1   .   .   .   .   .   .   . |
   |                                               |
 1 | >   .   .   .   1   .   .   .   .   .   .   1 |
   +-----------------------------------------------+
     1   2   3   4   5   6   7   8   9   10  11  12  13  AVE.
```

Karel sa pri riešení tejto úlohy môže spoliehať na nasledujúce podmienky:

- Karel začína na pozícii [1,1] a má v batohu dostatočné množstvo značiek.
- Stĺpy sú od seba vzdialené vždy 4 stĺpce, teda stĺpy sa postupne nachádzajú na 1, 5, 9 a na nasledujúcich Streets vo svete.
- Za všetkými stĺpmi sa na konci okamžite nachádza stena (v tomto príklade sa jedná o stĺpec č. 13).
- Na vrchu stĺpu je hneď stena, ale nemôžete si byť istí, že stĺpy sú rovnako vysoké.

Koncová situácia



- Niektoré stĺpy môžu obsahovať pôvodné kamene. Ak je to tak, váš program nesmie položiť značku na miesto, na ktorom sa už iná značka nachádza.

5. Vytvorte program, v ktorom bude robot Karel neustále prechádzať medzi dvoma protiľahlými stenami, pričom nezáleží na veľkosti sveta ani na polohe a orientácii robota. Úlohu riešte prostredníctvom nekonečného cyklu. Riešenie si môžete overiť na mapách walls1.kw a walls2.kw.

Poznámka Je bežné, že občas program uviazne. Pre ukončenie bežiaceho procesu v operačnom systéme Linux zvoľte CTRL+C.

6. Vytvorte pre robota Karla program, pomocou ktorého bude vedieť nájsť stred sveta, v ktorom sa nachádza. Karel sa na začiatku môže nachádzať na ľubovoľnej pozícii a môže byť taktiež otočený ľubovoľným smerom. Vo svete sa nenachádzajú žiadne steny ani žiadne značky a počet ulíc, ako aj počet avenues je nepárny. Riešenie úlohy si môžete overiť na mapách middleearth1.kw a middleearth2.kw.

Poznámka Podstatou tejto úlohy je precvičiť si algoritmické myslenie. Daný problém by ste preto nemali riešiť pomocou premenných (ak premenné už poznáte).

Poznámka Koncová situácia Vášho riešenia by nemala byť totožná so situáciou uvedenou na obrázku.

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (2, 3)  NORTH     99         0
ST.+--------------------+
 5 |.   .   .   .   .|
   |                  |
 4 |.   .   .   .   .|
   |                  |
 3 |.   ^   .   .   .|
   |                  |
 2 |.   .   .   .   .|
   |                  |
 1 |.   .   .   .   .|
   +------------------+
     1   2   3   4   5   AVE.
```

Koncová situácia

```
 12
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (3, 3)  WEST      99         0
ST.+--------------------+
 5 |.   .   .   .   .|
   |                  |
 4 |.   .   .   .   .|
   |                  |
 3 |.   .   <   .   .|
   |                  |
 2 |.   .   .   .   .|
   |                  |
 1 |.   .   .   .   .|
   +------------------+
     1   2   3   4   5   AVE.
```

3.5 Ďalšie zdroje

- PDCurses (http://pdcurses.codeforge.net/) - verzia knižnice *curses* pre operačný systém *Windows* (pdcurses.lib, pdcurses.dll)
- Knižnica *Karel the Robot* (Hlavičkový súbor, Windows, Linux 32b, Linux 64b)
- Rudolf Pecinovský: Základy algoritmizace (http://publikace.pecinovsky.cz/Zaklady_algoritmizace.pdf) - kapitola 10 (úvod), 10.2, 10.3, 11 (celá kapitola)
- Pavel Herout: Učebnice jazyka C (1. díl) (http://www.martinus.sk/?uItem=74741) - kapitoly 5.1, 5.4 a 5.5

4 Karel Becomes SuperKarel

4.1 Ciele

1. Osvojiť si návrh riešenia algoritmu pomocou techniky zhora-nadol.

4.2 Úvod

Úlohou tohto cvičenia je naučiť sa rozdeliť väčší problém na niekoľko menších a tieto potom postupne riešiť. Precvičíte si teda analýzu problému, jeho rozdelenie na menšie časti, návrh algoritmu a nakoniec celý problém naprogramujete.

4.3 Postup

Krok č. 1

Vytvorte pre robota Karla program, pomocou ktorého bude Karel vedieť zdvojnásobiť počet značiek, ktoré cestou nájde. Karel vždy začne na pozícii [1,1] a prejde celým svetom, pričom zdvojnásobí každú značku, ktorú nájde. Počet značiek, ktoré má Karel k dispozícii na riešenie tejto úlohy, je dostatočný.

Úloha 1.1

Vytvorte analýzu uvedeného problému. Pokúste sa celý problém rozdeliť na čo najmenšie funkčné celky (funkcie).

Úloha 1.2

Na základe analýzy vytvorte požadované funkcie, pomocou ktorých úlohu vyriešite.

Poznámka Podstatou tejto úlohy je precvičiť si algoritmické myslenie. Daný problém by ste preto nemali riešiť pomocou premenných (ak premenné už poznáte).

Poznámka	Pri riešení tejto úlohy je výhodné použiť rekurziu.

Overte svoje riešenie na nasledujúcich mapách: multiplier1.kw, multiplier2.kw.

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)   EAST      99         0
ST.+--------------------+
 5 | .   4   .   .   . |
   |                    |
 4 | .   .   .   .   . |
   |                    |
 3 | 4   3   .   .   1 |
   |                    |
 2 | .   .   .   .   . |
   |                    |
 1 | >   .   1   2   . |
   +--------------------+
     1   2   3   4   5   AVE.
```

Koncová situácia

```
 86
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (5, 5)   NORTH     84         0
ST.+--------------------+
 5 | .   8   .   .   ^ |
   |                    |
 4 | .   .   .   .   . |
   |                    |
 3 | 8   6   .   .   2 |
   |                    |
 2 | .   .   .   .   . |
   |                    |
 1 | .   .   2   4   . |
   +--------------------+
     1   2   3   4   5   AVE.
```

Krok č. 2

Robot Karel sa opäť vydal hľadať poklad. Tentokrát je jeho mapa plná značiek, ktoré mu napovedajú, ktorým smerom má ísť. Karel vždy začína na prázdnej pozícii a kráča rovno dovtedy, kým nenájde nejakú značku.

Úloha 2.1

Naprogramujte robota Karla tak, aby našiel poklad, ak pre mapu sveta platí:

- 1 značka znamená, že Karel musí ísť na sever,
- 2 značky znamenajú, že Karel musí ísť na západ,
- 3 značky znamenajú, že Karel musí ísť na juh,
- 4 značky znamenajú, že Karel musí ísť na východ,

- 5 značiek znamená, že Karel našiel poklad.
- Karel hľadá poklad, kým ho nenájde.

Poznámka Podstatou tejto úlohy je precvičiť si algoritmické myslenie. Daný problém by ste preto nemali riešiť pomocou premenných (ak premenné už poznáte).

Riešenie úlohy si môžete overiť na nasledujúcich mapách: treasuremap1.kw a treasuremap2.kw.

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (2, 2)  NORTH      0         0
ST.+---------------------------------------+
 8 | .  .  .  .  .  .  .  .  .  .  .  .  . |
   |                                       |
 7 | .  4  .  .  .  3  .  .  .  .  .  .  . |
   |                                       |
 6 | .  .  .  .  .  .  .  .  .  .  .  .  . |
   |                                       |
 5 | .  .  .  .  .  .  .  .  .  .  .  .  . |
   |                                       |
 4 | .  .  .  5  .  .  .  .  2  .  .  .  . |
   |                                       |
 3 | .  .  .  .  .  .  .  .  .  .  .  .  . |
   |                                       |
 2 | .  ^  .  .  .  4  .  .  1  .  .  .  . |
   |                                       |
 1 | .  .  .  .  .  .  .  .  .  .  .  .  . |
   +---------------------------------------+
     1  2  3  4  5  6  7  8  9  10  AVE.
```

Koncová situácia

```
 70
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (4, 4)  NORTH      0         0
ST.+---------------------------------------+
 8 | .  .  .  .  .  .  .  .  .  .  .  .  . |
   |                                       |
 7 | .  >  >  >  >  v  .  .  .  .  .  .  . |
   |                                       |
 6 | .  ^  .  .  .  v  .  .  .  .  .  .  . |
   |                                       |
 5 | .  ^  .  .  .  v  .  .  .  .  .  .  . |
   |                                       |
 4 | .  ^  .  >  <  <  <  <  <  <  .  .  . |
   |                                       |
 3 | .  ^  .  .  .  v  .  .  ^  .  .  .  . |
   |                                       |
 2 | .  ^  .  .  .  >  >  >  ^  .  .  .  . |
   |                                       |
 1 | .  .  .  .  .  .  .  .  .  .  .  .  . |
   +---------------------------------------+
     1  2  3  4  5  6  7  8  9  10  AVE.
```

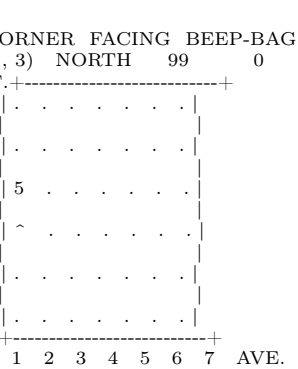
4.4 Doplňujúce úlohy

1. Vytvorte pre Karla príkaz stairs(), pomocou ktorého Karel postaví zo značiek schodisko. Na začiatku sa pred Karlom nachádza stĺpik postavený zo značiek, ktorých počet je väčší ako 1. Úlohou Karla je postaviť vpravo od tohto stĺpika schody reprezentované z ďalších značiek, pričom na každej pozícii bude vždy o jednu značku menej ako na predchádzajúcej. Karel má na začiatku dostatočný počet značiek, aby túto úlohu úspešne zvládol a vpravo od stĺpika je vždy dostatok miesta pre vytvorenie schodiska. Po vytvorení schodiska sa Karel bude nachádzať na jeho najvyššom stupni.

Poznámka Podstatou tejto úlohy je precvičiť si algoritmické myslenie. Daný problém by ste preto nemali riešiť pomocou premenných (ak premenné už poznáte).

Riešenie úlohy si môžete overiť na nasledujúcich mapách: stairsbuilder1.kw a stairsbuilder2.kw.

Počiatočná situácia



2. Vašou úlohou je upraviť riešenie stairs tak, aby poradie každého poschodia bolo označené odpovedajúcim počtom značiek. Karel vždy začína na prízemí a končí na najvyššom poschodí.

   Pôvodný program by ste mali upraviť tak, aby Karel očísloval schodisko s ľubovoľným počtom poschodí a zároveň tak, aby číslovanie schodiska bolo správne bez ohľadu na pôvodný počet značiek na každom poschodí.

Poznámka Podstatou tejto úlohy je precvičiť si algoritmické myslenie. Daný problém by ste preto nemali riešiť pomocou premenných (ak premenné už poznáte).

Koncová situácia

```
 36
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 4)  WEST      89         5
ST.+--------------------------+
 6 | .  .  .  .  .  .  .  . |
   |                        |
 5 | .  .  .  .  .  .  .  . |
   |                        |
 4 | <  4  3  2  1  .  .  . |
   |                        |
 3 | .  .  .  .  .  .  .  . |
   |                        |
 2 | .  .  .  .  .  .  .  . |
   |                        |
 1 | .  .  .  .  .  .  .  . |
   +--------------------------+
     1  2  3  4  5  6  7  AVE.
```

Poznámka	Pri riešení tejto úlohy je výhodné použiť rekurziu.

Riešenie úlohy si môžete overiť na nasledujúcich mapách: stairs1.kw a stairs2.kw.

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)  EAST      90         0
ST.+-----------------------------------+
 6 | .  .  .  .  .  .  .  .  .  . |
   |                              |
 5 | .  .  .  .  .  .  .  .  .  . |
   |              +---+           |
 4 | .  .  .  .  . | . | .  .  . |
   |           +---+   |          |
 3 | .  .  .  . | .   . | .  .  . |
   |        +---+       |          |
 2 | .  .  . | .  .  . | .  .  . |
   |     +---+          |          |
 1 | >  . | .  .  .  . | .  .  . |
   +-----------------------------------+
     1  2  3  4  5  6  7  8  9  AVE.
```

3. Upravte predchádzajúci program tak, aby Karlovi nezáležalo na výške jednotlivých schodov. Riešenie úlohy si môžete overiť na nasledujúcich mapách: stairs3.kw a stairs4.kw.

4. V spodnej rade Karlovho sveta sú ľubovoľne rozložené značky. Karel stojí na začiatku tejto rady (je doma). Vytvorte v rade nad ňou jej zrkadlový obraz. Napr. ak bude v pôvodnej rade značka (alebo značky) na ľavej krajnej pozícii, vo výslednej rade bude (budú) na pravej krajnej pozícii. Platí, že v spodnej rade Karlovho sveta sa na každej pozícii nachádza aspoň jedna značka. Riešenie úlohy si môžete overiť na mapách mirror1.kw a mirror2.kw.

Koncová situácia

```
167
CORNER  FACING  BEEP-BAG  BEEP-CORNER
(6, 5)  EAST    80        4
ST.+-----------------------------------+
 6 | .  .  .  .  .  .  .  .  .  . |
   |                               |
 5 | .  .  .  .  .  >  .  .  .  . |
   |              +---+            |
 4 | .  .  .  .  3 | . | .  .  .  . |
   |           +---+   |           |
 3 | .  .  .  2 | .  . | .  .  .  . |
   |         +---+     |           |
 2 | .  .  1 | .  .  . | .  .  .  . |
   |      +---+        |           |
 1 | .  . | .  .  .  . | .  .  .  . |
   +-----------------------------------+
     1  2  3  4  5  6  7  8  9  AVE.
```

Počiatočná situácia

```
0
CORNER  FACING  BEEP-BAG  BEEP-CORNER
(1, 1)  EAST    90        0
ST.+-----------------------------------+
 9 | .  .  .  .  .  .  .  .  .  . |
   |                               |
 8 | .  .  .  .  .  .  .  .  .  . |
   |                +---+          |
 7 | .  .  .  .  .  . | . | .  .  . |
   |                  |   |        |
 6 | .  .  .  .  .  . | . | .  .  . |
   |              +---+   |        |
 5 | .  .  .  .  1 | .  . | .  .  . |
   |            +---+     |        |
 4 | .  .  .  . | .  .  . | .  .  . |
   |            |         |        |
 3 | .  .  .  . | .  .  . | .  .  . |
   |         +---+         |        |
 2 | .  .  1 | .  .  .  . | .  .  . |
   |      +---+            |        |
 1 | >  . | .  .  .  .  . | .  .  . |
   +-----------------------------------+
     1  2  3  4  5  6  7  8  9  AVE.
```

Poznámka Podstatou tejto úlohy je precvičiť si algoritmické myslenie. Daný problém by ste preto nemali riešiť pomocou premenných (ak premenné už poznáte).

5. Upravte predchádzajúci program tak, aby v spodnej rade Karlovho sveta zostali pôvodne rozložené značky. Zároveň platí, že Karel má v batohu dostatočné množstvo značiek. Riešenie úlohy si môžete overiť na mapách mirror3.kw a mirror4.kw.

6. Upravte program so šachovnicou tak, aby nezáležalo na rozmeroch sveta (párnosť resp. nepárnosť políčok). Riešenie úlohy si môžete overiť na mapách empty6.kw, empty7.kw a empty8.kw.

Koncová situácia

```
 266
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (7, 8)  EAST      77         5
ST.+-----------------------------------+
 9 | .  .  .  .  .  .  .  .  .  . |
   |                               |
 8 | .  .  .  .  .  .  .  >  .  . |
   |                  +---+        |
 7 | .  .  .  .  .  . | . | .  . |
   |                  |   |      |
 6 | .  .  .  .  .  4 | . | .  . |
   |               +---+   |      |
 5 | .  .  .  .  3 | .  . | .  . |
   |           +---+       |      |
 4 | .  .  .  . | .  .  . | .  . |
   |            |         |      |
 3 | .  .  .  2 | .  .  . | .  . |
   |        +---+         |      |
 2 | .  .  1 | .  .  .  . | .  . |
   |     +---+            |      |
 1 | .  . | .  .  .  .  . | .  . |
   +-----------------------------------+
     1  2  3  4  5  6  7  8  9   AVE.
```

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)  EAST      0         1
ST.+---------------------------+
 3 | .  .  .  .  .  .  .  . |
   |                        |
 2 | .  .  .  .  .  .  .  . |
   |                        |
 1 | >  2  3  4  5  6  7  |
   +---------------------------+
     1  2  3  4  5  6  7   AVE.
```

Koncová situácia

```
 177
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 2)  WEST      0         7
ST.+---------------------------+
 3 | .  .  .  .  .  .  .  . |
   |                        |
 2 | <  6  5  4  3  2  1  |
   |                        |
 1 | .  .  .  .  .  .  .  . |
   +---------------------------+
     1  2  3  4  5  6  7   AVE.
```

Poznámka Podstatou tejto úlohy je precvičiť si algoritmické myslenie. Daný problém by ste preto nemali riešiť pomocou premenných (ak premenné už poznáte).

7. Prejdite na stránku http://kplab.tuke.sk/opravma (http://kplab.tuke.sk/opravma/). Na základe svojho loginu si nechajte vygenerovať zdrojový kód a opravte v

ňom všetky chyby, ktoré bránia tomu, aby bol výsledný kód úspešne preložený.

8. Vytvorte program, pomocou ktorého bude Karel vedieť vytvoriť vo svete, v ktorom sa nachádza, Fibonacciho postupnosť (http://en.wikipedia.org/wiki/Fibonacci_number). O svete platí, že je vysoký práve 2 riadky a široký n stĺpcov, pričom $n > 3$.

Počiatočná situácia

```
 0
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 1)  EAST      0         1
ST.+--------------------------------+
 2 | .   .   .   .   .   .   .   . |
   |                                |
 1 | >   .   .   .   .   .   .   . |
   +--------------------------------+
     1   2   3   4   5   6   7   8   AVE.
```

Koncová situácia

```
 556 TURNOFF
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (8, 1)  EAST      67        0
ST.+--------------------------------+
 2 | .   .   .   .   .   .   .   . |
   |                                |
 1 | 1   1   2   3   5   8   13  > |
   +--------------------------------+
     1   2   3   4   5   6   7   8   AVE.
```

4.5 Ďalšie zdroje

- PDCurses (http://pdcurses.codeforge.net/) - verzia knižnice *curses* pre operačný systém *Windows* (pdcurses.lib, pdcurses.dll)
- Knižnica *Karel the Robot* (Hlavičkový súbor, Windows, Linux 32b, Linux 64b)
- Rudolf Pecinovský: Základy algoritmizace (http://publikace.pecinovsky.cz/Zaklady_algoritmizace.pdf) - kapitoly 8, 10
- Pavel Herout: Učebnice jazyka C (1. díl) (http://www.martinus.sk/?uItem=74741) - kapitoly 5.1, 5.4, 5.5

5 Welcome to NetBeans!

5.1 Ciele

1. Oboznámiť sa so základnými údajovými typmi jazyka C.
2. Naučiť sa pracovať s premennými.
3. Jednoduché aritmetické výrazy
4. Zoznámiť sa s vývojovým prostredím NetBeans IDE a naučiť sa v ňom ladiť program.
5. Osvojiť si funkcie pre prácu so štandardným vstupom a výstupom.

5.2 Úvod

Na tomto cvičení sa naučíte pracovať s funkciami pracujúcimi so štandardným vstupom (čítanie z klávesnice) a štandardným výstupom (písanie na obrazovku). Tieto funkcie si osvojíte počas implementácie jednoduchých hier s názvom: *Koľko je hodín?* a *Uhádni číslo, ktoré si myslím.* Naučíte sa pracovať s vývojovým prostredím NetBeans IDE, ladiť v ňom programy, a nakoniec budete riešiť ďalšie Karlove problémy, pri riešení ktorých sa naučíte pracovať s premennými, ktoré Karlovi v mnohých prípadoch značne uľahčia prácu.

5.3 Postup

Krok č. 1 – Body Mass Index

V prvom kroku vytvoríte jednoduchý program na výpočet hodnoty BMI. Cieľom úloh je naučiť sa pracovať s prostredím *Netbeans*, s údajovými typmi, premennými a jednoduchými aritmetickými výrazmi v jazyku C.

Úloha 1.1

Vytvorte nový projekt v prostredí NetBeans IDE s názvom *BMI Counter*.

Pri vytváraní projektu vyberte možnosť vytvoriť *C/C++ Aplikáciu* a nezabudnite, že programujete v jazyku *C* a nie *C++*!

Úloha 1.2

Vytvorte funkciu countBmi(), ktorá vypočíta hodnotu BMI na základe hodnôt vstupných parametrov.

Funkcia bude mať nasledujúce vstupné parametre:

- height - výška v m, napr. 1.95,
- weight - hmotnosť v kg, napr. 95.2.

Výstupom funkcie bude hodnota vypočítaná podľa vzorca $BMI = (hmotnosvkg)/(vkavm)^2$.

Úloha 1.3

V hlavnej funkcii programu main() načítajte od používateľa hmotnosť a výšku a následne vypočítajte hodnotu BMI pomocou funkcie countBmi().

Príklad výstupu z programu je nasledovný, pričom vstup od používateľa je zvýraznený tučne a riadky začínajúce znakom '$' predstavujú príkazový riadok:

$ ./bmi

Zadaj výšku: 1.95

Zadaj hmotnosť: 95.2

Hodnota BMI je 25.0

$

Úloha 1.4

Rozšírte výstup z programu o vypísanie informácie, či sa na základe hodnoty BMI jedná o podvýživu (pod 18.5), normálnu hmotnosť (hodnota BMI medzi 18.5 a 24.9), nadváhu (hodnota BMI medzi 25 a 29.9) alebo obezitu (hodnota BMI nad 30).

Príklad výstupu z programu je nasledovný, pričom vstup od používateľa je zvýraznený tučne a riadky začínajúce znakom '$' predstavujú príkazový riadok:

$ ./bmi

Zadaj výšku: 1.95

Zadaj hmotnosť: 95.2

Hodnota BMI je 25.0, čo znamená: nadváha.

$

Krok č. 2 – Uhádni číslo, ktoré si myslím!

V tomto kroku vytvoríte jednoduchú hru s názvom *Guess the Number* alebo *Uhádni číslo, ktoré si myslím.*

Pravidlá hry sú veľmi jednoduché: počítač si vygeneruje náhodné číslo z určitého rozsahu, ktoré má hráč uhádnuť. Hráč sa potom pokúša tipovať, aké číslo si počítač myslí a ten mu zakaždým povie, či je jeho číslo väčšie, menšie alebo rovné s hráčovým tipom.

Cieľom nasledujúcich úloh je precvičiť si prácu so štandardným vstupom a výstupom, ako aj s jednoduchými výrazmi.

Úloha 2.1

Vytvorte nový projekt v prostredí NetBeans IDE s názvom *GuessTheNumber*.

Poznámka	Príručku pre prácu s prostredím NetBeans IDE nájdete v prílohe.

Úloha 2.2

Vytvorte program, ktorý vygeneruje náhodné číslo, načíta od hráča jeho tip a následne ho vyhodnotí správou na obrazovke.

Pre vygenerovanie náhodného čísla z intervalu <1, 100> v jazyku *C* použite nasledujúci fragment kódu:

```
1 srand(time(NULL));
2 int randomNumber = (rand() % 100) + 1;
```

Príklad výstupu z programu je nasledovný, pričom vstup od používateľa je zvýraznený tučne a riadky začínajúce znakom '$' predstavujú príkazový riadok:

$ ./guess

Myslím si číslo od 1 do 100.

Tvoj tip: 50

Hmm... Moje číslo je väčšie.

$

Úloha 2.3

Upravte svoje riešenie tak, aby hráč mohol číslo hádať dovtedy, pokiaľ ho neuhádne.

Príklad výstupu z programu je nasledovný, pričom vstup od používateľa je zvýraznený tučne a riadky začínajúce znakom '$' predstavujú príkazový riadok:

$ ./guess

Myslím si číslo od 1 do 100.

Tvoj tip: 50

Hmm... Moje číslo je väčšie.

Tvoj tip: 75

Hmm... Moje číslo je väčšie.

Tvoj tip: 87

Hmm... Moje číslo je menšie.

Tvoj tip: 81

Gratulujem! Uhádol si moje číslo.

$

Úloha 2.4

Obmedzte počet pokusov, ktoré môže hráč pri hádaní použiť, na 5 - ak ich vyčerpá, hráč číslo neuhádol.

V tomto momente budú existovať dva konce - úspešný a neúspešný. Úspešné ukončenie hry je identické s príkladom výstupu v predchádzajúcej úlohe. Ukážka neúspešného hádania je zobrazená nižšie, pričom vstup od používateľa je zvýraznený tučne a riadky začínajúce znakom '$' predstavujú príkazový riadok:

$ ./guess

Myslím si číslo od 1 do 100.

Tvoj tip: 50

Hmm... Moje číslo je väčšie.

Tvoj tip: 75

Hmm... Moje číslo je väčšie.

Tvoj tip: 87

Hmm... Moje číslo je menšie.

Tvoj tip: 81

Hmm... Moje číslo je väčšie.

Tvoj tip: 84

Hmm... Moje číslo je menšie.

Koniec hry. Moje myslené číslo je 82.

$

Svoje riešenie môžete rozšíriť aj o zobrazenie informácie o aktuálnom čísle pokusu.

Úloha 2.5

Rozšírte svoje riešenie tak, aby po skončení hádania čísla sa hra hráča opýtala, či si nechce zahrať znoa - ak áno, spustí hru znova; ak nie - program sa ukončí.

Príklad výstupu z programu je nasledovný, pričom vstup od používateľa je zvýraznený tučne a riadky začínajúce znakom '$' predstavujú príkazový riadok:

$ ./guess

Myslím si číslo od 1 do 100.

Tvoj tip: 50

Hmm... Moje číslo je väčšie.

Tvoj tip: 75

Gratulujem! Uhádol si moje číslo.

Chceš sa hrať ešte raz? (a/n): n

Tak ahoj nabudúce.

$

5.4 Doplňujúce úlohy

1. Modifikujte program *GuessTheNumber* tak, aby počítač hádal číslo, ktoré si myslíte vy ako hráč a vy mu pomocou odpovedí (moje je väčšie/menšie/rovné) budete pomáhať. Pre realizovanie programu využite metódu Bisekcie (http://en.wikipedia.org/wiki/Bisection_method) (Bisection search, metóda delenia intervalov).
   Príklad výstupu komunikácie z programu je nasledovný, pričom vstup od používateľa je zvýraznený tučne:

   Please think of a number between 0 and 100!

   Is your secret number 50?

   Enter 'h' to indicate the guess is too high. Enter 'l' to indicate the guess is too low. Enter 'c' to indicate I guessed correctly. l

   Is your secret number 75?

   Enter 'h' to indicate the guess is too high. Enter 'l' to indicate the guess is too low. Enter 'c' to indicate I guessed correctly. l

   Is your secret number 87?

   Enter 'h' to indicate the guess is too high. Enter 'l' to indicate the guess is too low. Enter 'c' to indicate I guessed correctly. h

   Is your secret number 81?

Enter 'h' to indicate the guess is too high. Enter 'l' to indicate the guess is too low. Enter 'c' to indicate I guessed correctly. l

Is your secret number 84?

Enter 'h' to indicate the guess is too high. Enter 'l' to indicate the guess is too low. Enter 'c' to indicate I guessed correctly. h

Is your secret number 82?

Enter 'h' to indicate the guess is too high. Enter 'l' to indicate the guess is too low. Enter 'c' to indicate I guessed correctly. l

Is your secret number 83?

Enter 'h' to indicate the guess is too high. Enter 'l' to indicate the guess is too low. Enter 'c' to indicate I guessed correctly. c

Game over. Your secret number was: 83

2. Modifikujte program *GuessTheNumber* nasledovne: Nech A si myslí číslo a B háda myslené číslo. Používateľ nech zadá interval, na ktorom je číslo hádané. Výstupom nech je konverzácia medzi A a B. Počítač teda bude komunikovať sám so sebou.
3. Vytvorte program, pomocou ktorého načítate zo vstupu postupnosť čísiel, pričom načítavanie ukončíte zadaním hodnoty 0. Na obrazovku vypíšte najväčšie zadané číslo, najmenšie zadané číslo, súčet všetkých zadaných čísiel a ich aritmetický priemer.
4. Vytvorte funkciu power(base,exp), ktorá vypočíta hodnotu na základe vzorca $base^e xp$. Úlohu môžete riešiť využitím rekurzie, ako aj iteratívnym postupom.
5. Vytvorte funkciu gcd(a,b), ktorá nájde najväčšieho spoločného deliteľa čísiel a a b. Napríklad:
   - gcd(2, 12) = 2
   - gcd(6, 12) = 6
   - gcd(9, 12) = 3
   - gcd(17, 12) = 1
6. Vytvorte funkciu fibonacci(n), ktorá na obrazovku vypíše prvých n členov fibonacciho postupnosti (http://en.wikipedia.org/wiki/Fibonacci_number). Úlohu môžete riešiť využitím rekurzie, ako aj iteratívnym postupom.
7. Vytvorte funkciu fact(n), ktorá vypočíta faktoriál zo zadanej hodnoty n. Úlohu môžete vyriešiť s použitím rekurzie aj bez nej.

5.5 Ďalšie zdroje

- www.netbeans.com (http://www.netbeans.com) - Domovská stránka prostredia *NetBeans*
- Cygwin (http://cygwin.com/) - Domovská stránka programu *Cygwin*
- PDCurses (http://pdcurses.codeforge.net/) - verzia knižnice *curses* pre operačný systém *Windows* (pdcurses.lib, pdcurses.dll)
- Knižnica *Karel the Robot* (Hlavičkový súbor, Windows, Linux 32b, Linux 64b)
- Rudolf Pecinovský: Základy algoritmizace (http://publikace.pecinovsky.cz/Zaklady_algoritmizace.pdf)
- Pavel Herout: Učebnice jazyka C (1. díl) (http://www.martinus.sk/?uItem=74741) - kapitoly 3.1 a 4.3

6 The World of Karel the Robot

6.1 Ciele

1. Osvojiť si prácu s jednorozmernými a dvojrozmernými poliami.
2. Tvorba vlastných makier.
3. Práca s príkazom switch.

6.2 Úvod

Zapnutie a vypnutie robota Karla, karlov svet - jeho načítanie do dvojrozmerného poľa a "vykreslenie"na obrazovku.

6.3 Postup

Krok č. 1 – Warmup!

Prvý krok predstavuje zahrievacie kolo, pretože si v ňom pripravíte prostredie pre vývoj. Vytvoríte si kostru projektu, na ktorom budete postupne počas cvičení pracovať a ďalej rozvíjať.

Úloha 1.1

Vo svojom vývojovom prostredí vytvorte nový projekt a v ňom okrem súboru main.c vytvorte aj súbory karel.c a karel.h.

Do súborov karel.c a karel.h budete písať všetky funkcie, premenné, makrá a vlastné údajové typy, ktoré súvisia s implementáciou knižnice robota Karla. Do súboru karel.c budete písať definície všetkých funkcií a potrebné globálne premenné a do súboru karel.h budete písať ich deklarácie, vlastné údajové typy a makrá.

Úloha 1.2

Nastavte si parametre prekladu vo svojom projekte tak, aby bol prekladaný nasledovne:

```
$gcc -std=gnu99 -Werror -pedantic
```

Poznámka Nezabudnite taktiež svoj projekt poslať do Git repozitáru (viď tento návod).

Úloha 1.3

Vytvorte makrá MAX$_{HEIGHT}$a`MAX_WIDTH`*aobenastavtenahodnotu*30.

Tieto makrá budú reprezentovať maximálnu šírku a maximálnu výšku sveta.

Poznámka V programovaní predstavuje menná konvencia (http://en.wikipedia.org/wiki/Naming_convention_(programming)#C_and_C.2B.2B_languages) súbor pravidiel pre pomenovávanie identifikátorov premenných, funkcií a pod. V rámci mennej konvencie sa pre pomenovávanie makier používajú výlučne LEN VEĽKÉ PÍSMENÁ.

Úloha 1.4

Definujte dve celočíselné premenné world$_{width}$a`world_height`, *ktorbudobsahovainformciuoaktulne*

Úloha 1.5

Vytvorte definíciu dvojrozmerného poľa s názvom world, ktorého šírku a výšku nastavte na hodnotu makier MAX$_{WIDTH}$a`MAX_HEIGHT`.

Obsahom poľa budú celočíselné hodnoty, ktoré budú reprezentovať jednotlivé prvky nachádzajúce sa vo svete robota Karla.

Úloha 1.6

Vytvorte makrá EMPTY a WALL, ktoré budú reprezentovať prvky sveta.

Hodnotu makra EMPTY nastavte na 0 a hodnotu makra WALL nastavte na -1.

Krok č. 2 – Stavanie sveta

Hlavnou úlohou tohto kroku bude vytvoriť funkcie turnOn(), turnOff() a draw(). Funkcia turnOn() bude slúžiť na zapnutie robota Karla a na načítanie sveta, v ktorom Karel bude vykonávať svoj program. Funkcia turnOff() zasa na jeho vypnutie. Funkcia draw() vykreslí na obrazovku svet, ktorého reprezentácia je uložená v premennej world.

Úloha 2.1

Vytvorte funkciu turnOn(), ktorá zo vstupu načíta svet robota Karla v zjednodušenom .kw formáte a uloží ho do dvojrozmerného poľa world.

Funkcia turnOn() nebude mať zatiaľ žiadne parametre a taktiež nebude vracať žiadnu hodnotu.

Príklad zjednodušeného .kw súboru je nasledovný:

```
5 6
W 3 1
W 2 2
W 3 2
W 4 2
W 3 3
W 2 4
W 3 4
W 4 4
.
```

Legenda:

- Prvý riadok obsahuje dve hodnoty: šírku a výšku načítavaného sveta.
- Nasledujúce riadky v tvare W X Y znamenajú pozíciu steny na súradniciach X a Y.
- Riadok, na ktorom je znak '.' ukončuje vstup.

Poznámka Pozor na problém pri načítavaní vstupných hodnôt pomocou funkcie scanf() - v buffri môže zostať neprečítaný koniec riadku. V prípade, že sa tak stane, použite nasledujúci snippet na jeho vyprázdnenie:

```
1 while (getchar() != '\n');
```

Úloha 2.2

Upravte svoje riešenie tak, aby sa načítavanie ukončilo v prípade zadania nekorektných údajov.

Za nekorektný vstup je možné požadovať nasledujúce situácie:

- načítaná hodnota šírky alebo výšky je väčšia ako hodnoty makier $\mathrm{MAX}_W IDTHa\mathtt{MAX_HEIGHT}, poprpadeichhoo$
- načítaný prvý znak (kód) je iný ako 'W', alebo
- umiestnenie steny je mimo mapu.

V prípade, že dôjde k jednej z uvedených situácií, vypíšte na obrazovku chybovú správu a program ukončite.

Poznámka Každý program, ktorý je ukončený, vracia do systému návratovú hodnotu. Táto hodnota reprezentuje jeho stav pri ukončovaní (viď Exit Status (http://en.wikipedia.org/wiki/Exit_status)). Pri ukončovaní vášho programu rozlišujte dva typy konca:

- Úspešný - program vtedy vráti hodnotu 0
- Neúspešný - program vtedy vráti hodnotu väčšiu ako 0

Miesto konkrétnych hodnôt môžete s výhodou použiť makrá EXIT$_S$*UCCESS*alebo`EXIT_FAILURE`z otandardnej kninice`stdlib.h`.

Úloha 2.3

Vytvorte funkciu draw(), ktorá vykreslí na obrazovku svet nachádzajúci sa v premennej world.

Funkcia nemá žiadny parameter a nevracia žiadnu hodnotu. Pri vykreslovaní sa riaďte nasledovne:

- ak v poli načítate hodnotu EMPTY, vypíšte na obrazovku medzeru (znak ' ') alebo (pre lepšiu orientáciu v mape) bodku (znak '.'),
- ak v poli načítate hodnotu WALL, vypíšte na obrazovku mriežku (znak '').

Úloha 2.4

Overte si správnosť svojho riešenia zavolaním oboch vytvorených funkcií priamo z funkcie main().

Poradie volania oboch funkcií je nasledovné:

```
1 turnOn();
2 draw();
```

V prípade úspechu sa na obrazovke vykreslí nasledujúci svet:

```
.....
.....
.###.
..#..
.###.
..#..
```

Úloha 2.5

Vytvorte funkciu turnOff(), ktorá slúži na korektné ukončenie práce s robotom Karlom.

V tele funkcie len vypíšte na obrazovku správu o autorovi programu.

6.4 Doplňujúce úlohy

1. Pridajte do Vášho riešenia komentáre a vygenerujte k nemu dokumentáciu vo formáte html prostredníctvom nástroja Doxygen.

Poznámka Základné informácie k nástroju Doxygen nájdete na jeho webovej stránke (http://www.doxygen.org), v pokynoch k vypracovaniu zadania č. 1 a v referenčnej príručke na stránkach predmetu Programovanie.

6.5 Ďalšie zdroje

- Rudolf Pecinovský: Základy algoritmizace (http://publikace.pecinovsky.cz/Zaklady_algoritmizace.pdf)
- Pavel Herout: Učebnice jazyka C (1. díl) (http://www.martinus.sk/?uItem=74741) - 11.1, 12.1, 12.2.3, 12.2.4, 13.1

7 Implementation of Karel the Robot

7.1 Ciele

1. Naučiť sa vytvárať funkcie, ktoré vracajú hodnoty.

7.2 Úvod

Na tomto cvičení vytvoríte funkcie movek() a turnLeft(), ktoré predstavujú základné primitívy pre robota Karla. Okrem nich vytvoríte taktiež senzory frontIsBlocked() a frontIsClear().

7.3 Postup

Krok č. 1

V prvom kroku vytvoríte niekoľko premenných, pomocou ktorých budete vedieť opísať Karlov aktuálny stav. Jedná sa o polohu a orientáciu robota Karla. Taktiež upravíte funkciu draw(), aby ste vedeli robota Karla správne vykresliť vzhľadom na jeho stav.

Úloha 1.1

Vytvorte premenné karel$_x$a`karel`$_y$, $ktorbudreprezentovax-ovay-ovsradnicuumiestneniarobotaK$

Úloha 1.2

Vytvorte premennú karel$_d$$irection, ktorbudereprezentovaorientciu, resp.smerrobotaKarla.$

Táto premenná môže nadobúdať iba nasledovné hodnoty:

- 0 - východ
- 90 - sever
- 180 - západ

- 270 - juh

Úloha 1.3

Aktualizujte funkciu turnOn() tak, aby na základe hodnôt zo vstupného formátu .kw správne rozpoznala pozíciu a smer robota Karla a nastavila podľa nich premenné karel_x, $\texttt{karel}_{\texttt{y}}a\texttt{karel}_{\texttt{d}}\texttt{irection}$.

Úloha 1.4

Upravte funkciu draw() tak, aby pri vykresľovaní sveta vykreslila na správnu pozíciu aj robota Karla.

Pre vykreslenie aktuálnej polohy robota Karla na obrazovku vzhľadom na jeho orientáciu, použite nasledujúce znaky:

- '>' - Karel je orientovaný na východ
- $'' - Karel je orientovan\acute{a}\ na\ sever$
- '<' - Karel je orientovaný na západ
- 'v' - Karel je orientovaný na juh

Úloha 1.5

Rozšírte funkciu draw() o stavový riadok, v ktorom vypíšete informáciu o aktuálnej pozícii a orientácii robota Karla.

Stavový riadok môže vyzerať napr. nasledovne:

CORNER FACING
(1, 2) East

alebo nasledovne:

POSITION = [1,2] - East

alebo ľubovoľne inak.

Úloha 1.6

Overte si správnosť svojej implementácie.

Svoju implementáciu si môžete overiť na nasledujúcom svete:

8 8 7 2 E
W 1 1
W 2 1
W 3 1
W 4 1

```
W 5 1
W 6 1
W 7 1
W 8 1
W 1 2
W 8 2
W 1 3
W 6 3
W 8 3
W 1 4
W 2 4
W 4 4
W 8 4
W 1 5
W 4 5
W 8 5
W 1 6
W 8 6
W 1 7
W 4 7
W 8 7
W 1 8
W 2 8
W 3 8
W 4 8
W 5 8
W 6 8
W 7 8
W 8 8
.
```

Výsledný svet spolu so stavovým riadkom bude potom vyzerať nasledovne:

```
CORNER  FACING
(7,2)   East
########
#  #   #
#      #
#  #   #
## #   #
#    # #
#     >#
########
```

Poznámka Pri reprezentácii sveta a robota Karla v ňom si dajte pozor na to, s akým počiatkom súradnicového systému budete pracovať. Pre nás ľudí je prirodzené pozerať sa na počiatok súradnicového systému (1,1) do ľavého dolného rohu. Preto pozíciu robota Karla budeme uvádzať vzhľadom na tento bod.

Krok č. 2

V tomto kroku implementujete funkciu turnLeft(), pomocou ktorej budete môcť robota Karla otočiť o 90 stupňov vľavo.

Úloha 2.1

Vytvorte funciu turnLeft(), pomocou ktorej otočíte Karla o 90 stupňov vľavo. Po otočení Karla rovno vykreslite zavolaním funkcie draw().

Pri otáčaní nezabudnite, že hodnoty Karlovej orientácie môžu byť len 0, 90, 180 a 270. Výsledok niektorých otočení preto bude potrebné upraviť.

Úloha 2.2

Overte si správnosť svojej implementácie.

Svoje riešenie si môžete overiť umiestnením robota Karla na ľubovoľné miesto na mape a jeho postupným otáčaním okolo svojej vlastnej osi. Overte taktiež, či sa Karel bude otáčať správnym smerom aj vtedy, ak sa budete otáčať okolo svojej osi viackrát.

Poznámka Pri opätovnom vykresľovaní sveta na obrazovku nedochádza k jeho prekresleniu, ale k vykresleniu scény nanovo pod predchádzajúcu scénu. Toto môže byť dosť mätúce najmä v prípade, ak máte malú mapu a veľkú obrazovku textovej konzoly, kde sa vám vojde niekoľko za sebou vykreslených krokov. Pre odstránenie tohto problému je možné napr.:

- vytvoriť vlastnú funkciu, ktorá vypíše na obrazovku dostatočný počet prázdnych riadkov, čím vznikne dojem jej vyčistenia; alebo
- zavolať funkciu system(), ktorá umožňuje spustiť príkaz z príkazového riadku (na zmazanie obrazovky je možné použiť príkaz cmd /c cls v operačnom systéme *Windows* a clear v operačnom systéme *Linux*).
- použiť na to určenú funkciu určitej špecifickej knižnice (napr. funkciu clear() knižnice curses.h)

Úloha 2.3

Vytvorte funkcie (senzory) facingNorth(), notFacingNorth(), facingSouth(), notFacingSouth(), facingEast(), notFacingEast(), facingWest() a notFacingWest().

Krok č. 3

V tomto kroku implementujete dva nové senzory pre robota Karla, a to frontIsClear() a frontIsBlocked(). Tieto senzory budete môcť neskôr použiť pre overenie karlovho pohybu - či sa môže presunúť na danú pozíciu alebo nie.

Úloha 3.1

Vytvorte funkciu frontIsClear(), ktorá vráti hodnotu 1 (true), ak sa Karel môže posunúť vpred. V opačnom prípade vráti hodnotu 0 (false).

Pri vytváraní funkcie dajte pozor na to, aby ste vedeli rozhodnúť vždy správne vzhľadom na orientáciu robota Karla. Senzor vráti hodnotu 0 (false) vtedy, ak je pred Karlom umiestnená stena alebo sa pred Karlom nachádza hranica sveta. V opačnom prípade vráti senzor hodnotu 1 (true).

Úloha 3.2

Vytvorte senzor frontIsBlocked(), ktorý bude opakom senzora frontIsClear().

Senzor bude vracať hodnotu 1 (true), ak sa Karel nemôže posunúť vpred. Ináč bude vracať hodnotu 0 (false).

Úloha 3.3

Overte si správnosť svojej implementácie.

Pre overenie správnosti implementácie začnite Karla postupne otáčať vľavo, až kým sa nedostanete do východzej polohy. Pri každom otočení si nechajte zobraziť výsledok senzorov frontIsClear() a frontIsBlocked().

Úloha 3.4

Podobne, ako ste vytvorili funkcie (senzory) frontIsClear() a frontIsBlocked() vytvorte aj funkcie leftIsClear(), leftIsBlocked(), rightIsClear() a rightIsBlocked().

Krok č. 4

V tomto kroku implementujete funkciu movek(), pomocou ktorej budete vedieť robota Karla posunúť o jeden krok vpred.

Úloha 4.1

Vytvorte funkciu movek(), ktorá posunie robota Karla o jeden krok vpred v prípade, že sa pred Karlom nenachádza žiadna stena alebo sa Karel nenachádza na okraji sveta.

Pre zistenie, či sa Karel môže alebo nemôže posunúť vpred, môžete s výhodou použiť vytvorený senzor frontIsClear(), resp. jeho variáciu frontIsBlocked().

Po úspešnom posunutí robota Karla ho rovno vykreslite zavolaním funkcie draw().

Ak sa Karel na danú pozíciu nemôže presunúť, vypíšte na obrazovku vhodnú správu a ukončite beh programu zavolaním funkcie exit(). Funkcia exit() pri svojom volaní požaduje parameter, ktorý reprezentuje tzv. *exit status* programu. Pre lepšiu prenositeľnosť kódu používajte nasledujúce hodnoty (makrá):

- $EXIT\_SUCCESS$ - toto makro použite vtedy, ak sa program ukončil úspešne (hodnota 0).
- $EXIT\_FAILURE$ - toto makro použite vtedy, ak sa program ukončil neúspešne (hodnota 1).

Poznámka Aby nedochádzalo k okamžitému vykresleniu robota Karla pri jeho posúvaní, môžete ho spomaliť zavolaním funkcie

- usleep() v operačnom systéme *Unix/Linux*, pričom jej deklarácia sa nachádza v hlavičkovom súbore unistd.h a jej parametrom je počet mikrosekúnd, alebo
- Sleep() v operačnom systéme *Windows*, pričom jej deklarácia sa nachádza v hlavičkovom súbore windows.h a jej parametrom je počet milisekúnd.

Úloha 4.2

Overte si správnosť svojej implementácie.

Pre overenie implementácie môžete použiť vybrané úlohy z predchádzajúcich cvičení.

7.4 Doplňujúce úlohy

1. Vytvorte počítadlo vykonaných krokov robota Karla. Za krok sa bude považovať len volanie funkcií movek() a turnLeft(). Počet krokov sa vypíše vždy spolu so zobrazením mapy (pri volaní funkcie draw()) v stavovom riadku.
2. Upravte zobrazenie počtu vykonaných krokov robota Karla tak, aby sa okrem informácie o počte krokov zobrazil aj názov posledne vykonaného kroku. Zobrazujte názvy iba týchto príkazov: movek(), turnLeft() a turnOn().
3. Pridajte do Vášho riešenia komentáre a vygenerujte k nemu dokumentáciu vo formáte html prostredníctvom nástroja Doxygen.

Poznámka Základné informácie k nástroju Doxygen nájdete na jeho webovej stránke (http://www.doxygen.org), v pokynoch k vypracovaniu zadania č. 1 a v referenčnej príručke na stránkach predmetu Programovanie.

4. Vytvorte funkciu strlen(char* str), ktorá vráti dĺžku reťazca str.
5. Vytvorte funkciu isNumber(char* str), ktorá zistí, či je reťazec číslom (vráti hodnotu 1) alebo nie (vráti hodnotu 0).

7.5 Ďalšie zdroje

- Rudolf Pecinovský: Základy algoritmizace (http://publikace.pecinovsky.cz/Zaklady_algoritmizace.pdf)
- Pavel Herout: Učebnice jazyka C (1. díl) (http://www.martinus.sk/?uItem=74741) - 5.6, 9.2, 12
- Knižnica ctype.h - Wikipédia (http://en.wikipedia.org/wiki/Ctype.h), Digital Mars (http://www.digitalmars.com/rtl/ctype.html) (spolu s príkladmi)
- Funkcia strlen() (http://www.cplusplus.com/reference/clibrary/cstring/strlen/) - Calculate the length of a string.

8 Karel and the Beepers

8.1 Ciele

1. Vytvárať funkcie, ktoré vracajú hodnoty.

8.2 Úvod

Na tomto cvičení bude vašou úlohou implementovať podporu značiek. Karel bude obsahovať batoh, do ktorého ich môže vkladať a na každej pozícii sveta bude možné umiestniť 0 a viac značiek. Okrem toho Karla vybavíme vhodnými senzormi, ktoré bude vedieť použiť na prácu so značkami.

8.3 Postup

Krok č. 1

V prvom kroku vytvoríte pre robota Karla batoh, v ktorom bude značky nosiť. Vytvoríte taktiež senzory beepersInBag() a noBeepersInBag(), pomocou ktorých bude Karel vedieť, či u seba nejakú značku má alebo nie.

Úloha 1.1

Vytvorte premennú karel$_b eepers, ktorbudereprezentovapoetznaiek, ktorKarelnesie.$

Úloha 1.2

Vytvorte senzor beepersInBag(), ktorý vráti hodnotu 1 (true), ak má Karel aspoň jednu značku v batohu. V opačnom prípade vráti hodnotu 0 (false).

Úloha 1.3

Vytvorte senzor noBeepersInBag(), ktorý bude opakom senzora beepersInBag().

Úloha 1.4

Aktualizujte stavový riadok doplnením o informáciu, koľko značiek sa nachádza v karlovom batohu.

Statový riadok môže po úprave vyzerať napr. takto:

CORNER FACING BEEP-BAG
(0, 0) North 10

alebo takto:

POSITION = [1,2] - East BEEP-BAG = 10

Úloha 1.5

Overte správnosť svojho riešenia inicializovaním premennej karel$_b$*eepersansledn*

Krok č. 2

V tomto kroku implementujete senzory beepersPresent() a noBeepersPresent(), pomocou ktorých bude Karel vedieť, či sa nachádza na rohu so značkami alebo nie. Taktiež aktualizujete funkciu draw(), v ktorej upravíte výpis stavového riadku a vykreslenie sveta.

Úloha 2.1

Vytvorte senzor beepersPresent(), ktorý vráti hodnotu 1 (true), ak sa na aktuálnej pozícii robota Karla nachádza aspoň jedna značka. V opačnom prípade vráti hodnotu 0 (false).

Počet značiek, ktoré sa budú nachádzať na mape, bude realizovaný kladnou hodnotou v dvojrozmernom poli reprezentujúcom mapu. To znamená, že ak bude hodnota na danej pozícii 8, počet značiek na danej pozícii bude tiež 8.

Úloha 2.2

Vytvorte senzor noBeepersPresent(), ktorý bude opakom senzora beepersPresent().

Úloha 2.3

Aktualizujte stavový riadok doplnením o informáciu, koľko značiek sa nachádza na aktuálnej pozícii robota Karla.

Statový riadok môže po úprave vyzerať napr. takto:

CORNER FACING BEEP-BAG BEEP-CORNER
(0, 4) North 10 0

alebo takto:

POSITION = [1,2]:0 - East BEEP-BAG = 10

Úloha 2.4

Aktualizujte funkciu turnOn() tak, aby pri načítavaní mapy sveta vedela správne rozpoznať aj rozmiestnenie značiek v ňom.

Ak by sa v mape na pozícii [x,y] malo nachádzať z značiek, súbor sveta musí obsahovať nasledujúci riadok:

B x y z

Úloha 2.5

Aktualizujte funkciu draw() tak, aby ste do mapy sveta vniesli informáciu o značkách, ktoré sa v ňom môžu nachádzať.

Keďže svet bude v tejto implementácii dosť "úzky", nie je potrebné zobrazovať počet značiek na konkrétnej pozícii. Pokiaľ si ho ale vhodne rozšírite (zväčšením vzdialeností medzi susednými pozíciami), môžete. Stačí však, ak miesto, na ktorom sa značka nachádza, zvýrazníte špeciálnym znakom (v našom prípade budeme používať značku hviezdička, $'*'$).

Úloha 2.6

Overte správnosť svojho riešenia napísaním funkcie walk(), pomocou ktorej robot Karel prejde postupne celý svet.

Pre overenie karlovej prechádzky môžete použiť nasledujúci svet:

```
6 5 1 1 E 0
B 1 1 1
B 1 2 1
B 1 3 1
B 1 4 1
B 1 5 1
B 2 1 1
B 2 2 1
B 2 3 1
B 2 4 1
B 2 5 1
B 3 1 1
B 3 2 1
B 3 3 1
B 3 4 1
B 3 5 1
B 4 1 1
```

```
B 4 2 1
B 4 3 1
B 4 4 1
B 4 5 1
B 5 1 1
B 5 2 1
B 5 3 1
B 5 4 1
B 5 5 1
B 6 1 1
B 6 2 1
B 6 3 1
B 6 4 1
B 6 5 1
.
```

Počiatočná situácia

```
CORNER  FACING BEEP-BAG BEEP-CORNER
(0, 0)  East     0        0

******
******
******
******
>*****
```

Koncová situácia

```
CORNER  FACING BEEP-BAG BEEP-CORNER
(5, 4)  North    0        1

*****^
******
******
******
.*****
```

Poznámka Ako ste si všimli, svet nemusí mať na všetkých krajných pozíciách stenu. Tomu je potrebné prispôsobiť aj funkcie pre senzory. Ak vaše riešenie zatiaľ podobné situácie neuvažuje, vráťte sa k Úlohe 3.1 z predošlého cvičenia.

Krok č. 3

V tomto kroku vytvoríte dve funkcie, ktoré budú reprezentovať dva karlove príkazy: funkcia putBeeper(), pomocou ktorej Karel položí z batohu značku

na aktuálnu pozíciu; a funkcia pickBeeper(), pomocou ktorej Karel vezme z aktuálnej pozície značku a vloží si ju do batohu.

Úloha 3.1

Vytvorte funkciu putBeeper(), pomocou ktorej Karel položí z batohu značku na aktuálnu pozíciu.

Pri implementácii ošetrite prípad, keď Karel v batohu nemá žiadnu značku. Vtedy ukončite program s vypísaním vhodnej chybovej správy.

Úloha 3.2

Vytvorte funkciu pickBeeper(), pomocou ktorej Karel vezme značku z aktuálnej pozície a vloží si ju do batohu.

Pri implementácii ošetrite prípad, keď sa na aktuálnej karlovej pozícii nenachádza žiadna značka. Vtedy ukončite program s vypísaním vhodnej chybovej správy.

Poznámka Pri volaní oboch funkcií nezabudnite aktualizovať stavový riadok zavolaním funkcie draw()!

Úloha 3.3

Overte správnosť svojho riešenia napísaním funkcie harvestAll(), pomocou ktorej robot Karel prejde postupne celý svet a pozbiera v ňom všetky značky. Po pozbieraní Karel všetky pozbierané značky položí a vráti sa do východzej pozície.

Pre riešenie úlohy môžete použiť predchádzajúcu mapu (fullOfStars).

Počiatočná situácia

```
CORNER   FACING  BEEP-BAG  BEEP-CORNER
(0, 0)    East       0         0

******
******
******
******
>*****
```

8.4 Doplňujúce úlohy

1. Vytvorte funkciu transformation(), ktorá prevedie formát sveta pre robota Karla, ktorý bol používaný v knižnici karel v prvých cvičeniach, na formát, ktorý je používaný teraz. Funkcia nebude mať žiadny parameter, ale bude

Koncová situácia

```
CORNER   FACING  BEEP-BAG  BEEP-CORNER
(0, 0)    South      0         0

.....*
......
......
......
v.....
```

vracať reťazec znakov (char*), ktorý bude reprezentovať výslednú prevedenú mapu.

V prípade načítavania značiek kvôli novému formátu stačí rozpoznať, či sa na danej pozícii značka nachádza alebo nie. Ak sa teda v pôvodnej mape bude nachádzať na pozícii 10 značiek, do novej mapy túto informáciu prevediete len s hodnotou 1.

Načítanie sveta vo formáte použitom v knižnici karel vykonajte ručne. To znamená, že svet budete načítavať priamo zo vstupu postupným zadávaním jednotlivých hodnôt.

Poznámka V operačnom systéme *Linux* môžete miesto ručného zadávania hodnôt s výhodou použiť presmerovanie vstupu, napr.:

```
1 $ ./a.out < empty.kw
```

Funkciu volajte z funkcie turnOn() vtedy, ak parameter funkcie turnOn() bude obsahovať hodnotu $NULL$.

Príklad vstupu:

5 4 1 1 E 10
B 2 3 10
W 3 3

Vysvetlenie: Svet má rozmer 5x4, Karel sa na začiatku nachádza na pozícii [1,1], je otočený na východ, má v batohu 10 značiek, na pozícii [2,3] sa nachádza 10 značiek a na pozícii [3,3] je stena.

2. Pridajte do Vášho riešenia komentáre a vygenerujte k nemu dokumentáciu vo formáte html prostredníctvom nástroja Doxygen.

Poznámka Základné informácie k nástroju Doxygen nájdete na jeho webovej stránke (http://www.doxygen.org), v pokynoch k vypracovaniu zadania č. 1 a v referenčnej príručke na stránkach predmetu Programovanie.

8.5 Ďalšie zdroje

- Rudolf Pecinovský: Základy algoritmizace (http://publikace.pecinovsky.cz/Zaklady_algoritmizace.pdf)
- Pavel Herout: Učebnice jazyka C (1. díl) (http://www.martinus.sk/?uItem=74741) - 9.2

9 Karel and his Sensors

9.1 Ciele

1. Porozumieť a vedieť využiť vo svojich programoch štruktúrovaný typ záznam a údajové objekty tohoto typu.
2. Porozumieť a vedieť využiť vo svojich programoch enumeračné typy.

9.2 Úvod

Doposiaľ ste pracovali len s jednoduchými a základnými (preddefinovanými) typmi. Na dnešnom cvičení sa naučíte vytvárať vlastné štruktúrované typy.

9.3 Postup

Krok č. 1

V prvom kroku nahradíte niekoľko premenných, ktoré opisujú robota Karla, prostredníctvom štruktúry záznam.

Úloha 1.1

Vytvorte štruktúru záznam s názvom Robot, ktorá bude zoskupovať všetky premenné opisujúce vlastnosti robota Karla.

Štruktúra Robot bude predstavovať zoskupenie premenných karel$_x$, karel$_y$, karel$_b$eepers$a$karel$_d$i

> **Poznámka** Pokiaľ ste pracovali aj na doplňujúcich úlohách, do štruktúry Robot zahrňte aj premenné karel$_s$$tepsapremenn, vktorejuchovvatenzovposlednevykonanhoprkazu.$

Úloha 1.2

Definujte premennú karel, ktorá bude typu Robot.

Úloha 1.3

Aktualizujte svoj kód nahradením pôvodných premenných za vytvorenú štrukturálnu premennú.

Krok č. 2

V tomto kroku vytvoríte podobne, ako tomu bolo v prvom kroku, samostatnú štruktúru záznam zoskupujúcu premenné opisujúce svet robota Karla.

Úloha 2.1

Vytvorte štruktúru záznam s názvom World, ktorá bude zoskupovať všetky premenné opisujúce svet, v ktorom sa bude robot Karel nachádzať.

Štruktúra World bude predstavovať zoskupenie premenných world$_h$*eight*, `world`$_{\mathtt{w}}$`idth`*a*`world`.*Nzvypoloiektruk*

Úloha 2.2

Definujte premennú world, ktorá bude typu World.

Úloha 2.3

Aktualizujte svoj kód nahradením pôvodných premenných za vytvorenú štrukturálnu premennú.

Krok č. 3

V tomto kroku využijete vo svojej implementácii vlastné enumeračné typy Direction a Boolean.

Úloha 3.1

Vytvorte enumeračný typ Direction, ktorý bude reprezentovať možné smery, v ktorých sa Karel môže pohybovať.

Pri vytváraní nezabudnite jednotlivé položky (EAST, NORTH, WEST a SOUTH) enumeračného typu inicializovať na správne hodnoty.

Úloha 3.2

Aktualizujte svoj kód použitím enumeračného typu Direction vo svojej implementácii.

Všetky číselné konštanty, ktoré používate na reprezentáciu smerov, nahraďte položkami enumeračného typu. Taktiež všetky premenné, ktoré nesú informáciu o smere (orientácii) robota Karla, pretypujte na typ Direction.

Úloha 3.3

Vytvorte enumeračný typ Boolean, ktorý bude mať len dve hodnoty: TRUE (pravda, hodnota 1) a FALSE (nepravda, hodnota 0).

Úloha 3.4

Aktualizujte svoj kód použitím enumeračného typu Boolean vo svojej implementácii.

Aktualizácia sa bude týkať najmä všetkých senzorov - ich návratového typu, ako aj porovnávaní výstupnej hodnoty zo senzoru s konkrétnou číselnou hodnotou (1 alebo 0).

Krok č. 4

V poslednom kroku implementujete zvyšné senzory robota Karla. Ich prehľad ako aj význam nájdete na tejto linke.

Úloha 4.1

Postupne implementujte zvyšné senzory robota Karla.

Úloha 4.2

Overte správnosť svojho riešenia overením fungovania jednotlivých senzorov.

9.4 Doplňujúce úlohy

1. Robot Karel nesmie vykonať žiadny príkaz, pokiaľ nie je zapnutý. Upravte svoju implementáciu tak, aby primitívy ako aj samotné senzory robota Karla nebolo možné použiť skôr, ako bol robot Karel zapnutý pomocou funkcie turnOn().
2. Pridajte do Vášho riešenia komentáre a vygenerujte k nemu dokumentáciu vo formáte html prostredníctvom nástroja Doxygen.

Poznámka Základné informácie k nástroju Doxygen nájdete na jeho webovej stránke (http://www.doxygen.org), v pokynoch k vypracovaniu zadania č. 1 a v referenčnej príručke na stránkach predmetu Programovanie.

9.5 Ďalšie zdroje

- Rudolf Pecinovský: Základy algoritmizace (http://publikace.pecinovsky.cz/Zaklady_algoritmizace.pdf)
- Pavel Herout: Učebnice jazyka C (1. díl) (http://www.martinus.sk/?uItem=74741) - 14.1, 14.2

10 Sokoban Intermezzo: the Curses

10.1 Ciele

1. Osvojiť si základy práce s knižnicou curses.

10.2 Úvod

Výpis na pozíciu, odchytávanie stlačených kláves, farebný výstup.

Na tomto cvičení začínate pracovať na svojom druhom zadaní - jednoduchej konzolovej hre Sokoban (http://en.wikipedia.org/wiki/Sokoban). Pri práci môžete vychádzať z vašej implementácie knižnice robota Karla a pri čítaní nasledujúcich modulov budú jednotlivé úlohy nadväzovať na vašu predchádzajúcu implementáciu.

Hlavným cieľom tohto cvičenia je však zoznámiť sa s knižnicou curses, ktorá vám poskytne viac možností pri práci s obrazovkou, ako poskytujú funkcie printf() alebo putchar(). Okrem toho umožňuje odchytiť stlačenie klávesy, čo budete vedieť využiť pri ovládaní svojej postavy v hre.

10.3 Postup

Krok č. 1

Ak chceme pracovať s knižnicou curses, je potrebné ju na začiatku programu inicializovať a po skončení behu programu zasa obnoviť pôvodné nastavenia. V opačnom prípade totiž nebude možné používať jednotlivé funkcie knižnice. Preto v tomto kroku upravíte funkciu turnOn() tak, aby inicializovala režim knižnice curses a vo funkcii turnOff() zasa tento režim vypnete.

Úloha 1.1

Pripojte do svojho programu knižnicu curses.h a vyriešte všetky vzniknuté problémy.

Jeden z možných problémov sa bude týkať vlastného typu boolean, ktorý ste vytvárali. Typ boolean sa totiž v knižnici curses nachádza.

Poznámka Pokiaľ pracujete na *OS Windows*, môže sa vám pri preklade zobraziť varovanie v podobe "MOUSE$_{M}OVED$" $redefined. Abystesamuvyhli, vlotekninicu$`curses.h`$predkninicu$`windows.h`.

Úloha 1.2

Vo funkcii turnOn() inicializujte režim knižnice curses pomocou funkcie initscr().

Úloha 1.3

Upravte funkciu turnOff(), ktorá ukončí režim práce knižnice curses a obnoví tak pôvodné nastavenia terminálu.

Úloha 1.4

Overte správnosť svojej implementácie.

Poznámka Ak ste pracovali v OS *Linux* a postupovali ste správne, tak sa na obrazovke nič nezobrazí. Je to spôsobené tým, že knižnica curses pracuje vo vlastnom režime a pre výpis na obrazovku nepoužíva funkciu printf(), ktorú zatiaľ vo svojom programe používate.

Krok č. 2

Keďže sa aktuálne na obrazovku nevykreslí scéna, ktorú už máte pripravenú, v tomto kroku aktualizujete funkciu draw() tak, aby bola kompatibilná s knižnicou curses.

Úloha 2.1

Aktualizujte funkciu draw() tak, aby vedela vykresliť mapu na obrazovku s použitím funkcií knižnice curses.

Pre zvládnutie tejto úlohy budete potrebovať poznať nasledujúce funkcie:

- printw() - print formatted output in curses windows
- move() - move curses window cursor
- refresh() - refresh curses windows and lines
- clear() - clears the screen

Poznámka Pri používaní funkcie move() nezabudnite, že pozícia $(0, 0)$ sa nachádza v ľavom hornom rohu.

Úloha 2.2

Overte správnosť svojej implementácie.

Krok č. 3

Zatiaľ viete hráča ovládať len pomocou volania funkcií, ktoré reprezentujú primitívy robota Karla. V tomto kroku využijete funkcie nachádzajúce sa v knižnici curses na to, aby ste mohli hráča ovládať aj pomocou klávesnice.

Úloha 3.1

Vo funkcii main() vytvorte nekonečnú slučku a načítavajte v nej kódy stlačených kláves pomocou volania funkcie getch(). Zo slučky (programu) vyškočte stlačením klávesy **q** .

Poznámka Pokiaľ overíte funkčnosť svojho riešenia teraz, každý stlačený kláves sa rovno zobrazí na obrazovke. Pri hraní hier to však nie je vhodné. Ak sa chcete tomuto efektu vyhnúť, použite funkciu noecho(). Zobrazovanie znakov opäť zapnete zavolaním funkcie echo().

Úloha 3.2

Namapujte stlačenie klávesy **KEY**$_U$ *Pnavolaniefunkcie*`movek`().

Poznámka Aby ste mohli pracovať aj s kurzorovými klávesmi, je potrebné zapnúť *keypad* na termináli používateľa. To je možné vykonať zavolaním funkcie keypad(), ktorej parametrom okna (win) bude konštanta *stdscr* reprezentujúca štandardný výstup.

Úloha 3.3

Namapujte stlačenie klávesy KEY$_L$*EFTnavolaniefunkcie*`turnLeft`().

Úloha 3.4

Overte správnosť svojej implementácie.

10.4 Doplňujúce úlohy

1. Oboznámte sa s podporou a prácou s farbami v knižnici curses a vhodným spôsobom ich použite vo svojom riešení. Pre prácu s farbami môžete použiť nasledujúce funkcie:

- has$_c olors()-vracia$TRUE$alebo$FALSE$podatoho, iterminlpodporujeprcusfarbami$
- start$_c olor()-funkciamusbyzavolan, akchcetevprogramepouvafarby(idelnerovnopofunkcii$
- init$_p air()-mendefinciudvojicefarieb, priomdvojicafariebjechpanakofarbapozadiaafarba$ $slopru, islofarbypoprediaaslofarbypozadia.$
- attrset() - nastaví atribúty okna na atribúty zadané ako argument funkcie, pričom parametrom môže byť aj dvojica farieb definovaná funkciou init$_p air()$.

2. Upravte funkciu draw() tak, aby sa pri každom volaní nemusela vykreslovať celá scéna znova, ale prekreslili sa len vzniknuté zmeny. Za zmenu sa v tomto prípade dá považovať stará a nová pozícia robota Karla, ako aj aktualizácia stavového riadku.
3. V prípade, že sa s robotom Karlom nechcete rozlúčiť tak skoro, tak namapujte vhodné klávesové skratky aj na funkcie putBeeper() a pickBeeper().

10.5 Ďalšie zdroje

- curses (programming library) (http://en.wikipedia.org/wiki/Curses_(programming_library))
- PDCurses (http://pdcurses.codeforge.net/) - verzia knižnice *curses* pre operačný systém *Windows* (pdcurses.lib, pdcurses.dll)
- Writing Programs with NCURSES (http://invisible-island.net/ncurses/ncurses-intro.html)
- Introduction to the Unix Curses Library (http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Curses.pdf)
- initscr(), endwin() (http://www.mkssoftware.com/docs/man3/curs_initscr.3.asp) - curses screen initialization and manipulation routines
- getch() (http://www.mkssoftware.com/docs/man3/curs_getch.3.asp) - get characters from curses terminal keyboard
- refresh() (http://www.mkssoftware.com/docs/man3/curs_refresh.3.asp) - refresh curses windows and lines
- printw() (http://www.mkssoftware.com/docs/man3/curs_printw.3.asp) - print formatted output in curses windows
- move() (http://www.mkssoftware.com/docs/man3/curs_move.3.asp) - move curses window cursor
- echo(), noecho() (http://www.mkssoftware.com/docs/man3/curs_inopts.3.asp) - curses input options
- curs_set() (http://www.mkssoftware.com/docs/man3/curs_kernel.3.asp) - sets the cursor state to invisible, normal, or very visible for visibility equal to 0, 1, or 2 respectively. If the terminal supports the visibility requested, the previous cursor state is returned; otherwise, ERR is returned.

- keypad() (http://www.mkssoftware.com/docs/man3/curs_inopts.3.asp) - enables the keypad of the user's terminal. If enabled (bf is TRUE), the user can press a function key (such as an arrow key) and getch() returns a single value representing the function key, as in $\mathbf{KEY}_LEFT$.
- has_colors(), start_color(), init_pair() (http://www.mkssoftware.com/docs/man3/curs_color.3.asp) - curses color manipulation routines
- attrset() (http://www.mkssoftware.com/docs/man3/curs_attr.3.asp) - sets the current attributes of the given window to attrs
- clear() (http://www.mkssoftware.com/docs/man3/curs_clear.3.asp) - clears the screen

11 Sokoban - transform!

11.1 Ciele

1. Pracovať s knižnicou curses.
2. Využiť predchádzajúce vedomosti pri implementácii riešenia.

11.2 Úvod

Na tomto cvičení budete pokračovať v práci s knižnicou curses. Tiež pretransformujete robota Karla na Sokobana.

11.3 Postup

Krok č. 1

V prvom kroku vykonáte všetky potrebné kroky, ktoré sú potrebné pre pretransformovanie aktuálnej verzie vašej implementácie robota Karla na *Sokobana*. Zmeny sa týkajú klávesových skratiek, ako aj nových prvkov, ktoré sa môžu v miestnosti zobraziť. Taktiež správanie niektorých existujúcich prvkov bude zmenené.

Úloha 1.1

Na klávesy $\mathbf{KEY}_{D}OWNa\mathbf{KEY}_{\mathbf{R}}\mathbf{IGHT}namapujtepohybdoluapohybvpravoapremapujteklvesu\mathbf{K}$

Sokoban sa bude pohybovať priamo o pozíciu vpred/vzad/vľavo/vpravo. Nebude teda potrebné mať samostatnú klávesovú skratku pre jeho otočenie vľavo alebo vpravo. Samozrejme ale na otočenie sokobana môžete použiť funkciu turnLeft() **a následne volať funkciu** movek()**.**

Ak ste vo svojej implementácii počítali kroky, ktoré spravil robot Karel, aktualizujte taktiež svoj kód tak, aby rátal všetky kroky vpred/vzad/vľavo/vpravo, ale nerátal za krok otočenie sokobana vľavo.

Úloha 1.2

Aktualizujte funkciu turnOn() **tak, aby vedela rozpoznávať aj znaky** '*', '.' **a** '$'.

Význam uvedených znakov je nasledovný:

- **znak** '$' **reprezentuje krabicu (box), ktorú má Sokoban presunúť na cieľové miesto**
- **znak** '.' **reprezentuje miesto, kam má Sokoban dotlačiť krabicu (box)**
- **znak** '*' **reprezentuje krabicu, ktorá sa nachádza na cieľovom mieste**

Pre ich reprezentáciu môžete opäť vhodne využiť vlastné makrá.

Úloha 1.3

Aktualizujte funkciu movek() **tak, aby Sokoban vedel posúvať krabice, ktoré sa pred ním môžu nachádzať.**

Ak sa pred Sokobanom bude nachádzať krabica, bude ju Sokoban vedieť posunúť o jednu pozíciu vpred.

Počiatočná situácia

```
#####
#   #
# $ #
# ^ #
#####
```

Koncová situácia

```
#####
# $ #
# ^ #
#   #
#####
```

Nezabudnite pri svojej implementácii taktiež na to, že krabica sa môže presunúť na cieľovú pozíciu alebo ju môžete presunúť z cieľovej pozície. V tomto prípade dôjde len k zmene jej zobrazenia vo funkcii draw()**.**

Počiatočná situácia

```
#####
# . #
# $ #
# ^ #
#####
```

Sokoban vie naraz posunúť len jednu krabicu. Ak sa teda pred Soko-

Koncová situácia

```
#####
# * #
# ^ #
#   #
#####
```

banom budú nachádzať dve krabice súčasne, nepohne ani s jednou z nich.

Počiatočná situácia

```
#######
#     #
# $$< #
#######
```

Koncová situácia

```
#######
#     #
# $$< #
#######
```

Úloha 1.4

Otestujte správnosť svojej implementácie na nasledujúcej mape.

```
1 char* map = "---####|---#--#|---#--#|---#$
      .#|####--#|#-----#|#@$.-#|########";
```

Grafická reprezentácia uvedenej mapy je nasledujúca:

```
  ####
  #  #
  #  #
  #$.#
####  #
#     #
#>$.  #
#######
```

Krok č. 2

V tomto kroku aktualizujete stavový riadok a taktiež zareagujete na aktuálny stav hry.

Úloha 2.1

Rozšírte stavový riadok o informáciu ohľadom celkového počtu krabíc, ktoré sa v miestnosti nachádzajú a taktiež o informáciu, koľko krabíc ešte zostáva presunúť alebo koľko ich už bolo presunutých na cieľové miesto.

Stavový riadok môže vyzerať nasledovne:

```
CORNER  FACING  DELIVER STEPS
(1,6)   East      0/2     0
```

čo v tomto prípade znamená, že na cieľovom mieste sa ešte nenachádza žiadna krabica.

Úloha 2.2

Po presunutí všetkých krabíc na cieľové miesto vypíšte na obrazovku správu informujúcu o úspešnom ukončení hry a hru ukončite.

Počiatočná situácia

```
CORNER  FACING  DELIVER STEPS
(1,6)   East    2/2     0
  ####
  #  #
  #  #
  #$.#
####  #
#     #
#>$.  #
#######
```

Koncová situácia

```
CORNER  FACING  DELIVER STEPS
(2,6)   East    0/2     29
  ####
  #  #
  #  #
  # *#
####  #
#     #
# >*  #
#######
Well done
```

11.4 Doplňujúce úlohy

1. Pridajte do hry rolujúci text (horizontálny, v smere zprava doľava), ktorý sa bude bežať počas celej hry v najspodnejšom riadku hry, poprípade rolujúci text môže bežať iba pri zobrazenom menu, ktoré budete implementovať budúce cvičenie.
2. Po ukončení hry zobrazte na obrazovke záverečné kredity. Efekt zobrazovania

si môžete zvoliť sami (napr. vo forme filmových tituliek zdola nahor alebo postupným zobrazovaním jednotlivých správ alebo ľubovoľný iný).

11.5 Ďalšie zdroje

- Sokoban (http://en.wikipedia.org/wiki/Sokoban) (Wikipédia)
- PDCurses (http://pdcurses.codeforge.net/) - verzia knižnice *curses* pre operačný systém *Windows* (pdcurses.lib, pdcurses.dll)
- Writing Programs with NCURSES (http://invisible-island.net/ncurses/ncurses-intro.html)
- Introduction to the Unix Curses Library (http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Curses.pdf)

12 Sokoban levels

12.1 Ciele

1. Modulárne programovanie.
2. Dynamické údajové typy.
3. Správa pamäte.
4. Práca so súbormi.

12.2 Úvod

Hru samotnú robí zaujímavou to, že po úspešnom vyriešení jedného problému dostane hráč na riešenie problém ďalší a hlavne ťažší. Hra je teda tvorená súborom niekoľkých levelov (alebo úrovní), ktoré musí hráč zdolať. Ak ich všetky úspešne zdolá, hru samotnú dohrá.

Na tomto cvičení vytvoríte vo svojej hre mechanizmus, pomocou ktorého bude hráč môcť prechádzať z levelu do levelu. Samostatné levely budú uložené v externom súbore a pri spustení hry sa zavedú do pamäte, v ktorej budú reprezentované pomocou spojkového zoznamu.

12.3 Postup

Krok č. 1

V prvom kroku rozdelíte celý projekt na niekoľko samostatných častí - modulov, do ktorých potom popresúvate jednotlivé časti svojho kódu. Rozdelenie kódu na menšie časti (moduly) zvyšuje jeho prehľadnosť a čitateľnosť.

Úloha 1.1

Vytvorte v projekte hlavičkový súbor game.h a k nemu prislúchajúci súbor

game.c a umiestnite do nich všetky "knižničné"funkcie, ich deklarácie a definície, ako aj všetky príslušné makrá a štruktúry.

Úloha 1.2

Aktualizujte súbor main.c tak, aby používal hlavičkový súbor game.h a volal funkcie, ktoré sa v ňom nachádzajú.

Úloha 1.3

Overte správnosť svojej úpravy.

Správnosť overíte prekladom takto upraveného projektu, ktorý by mal byť v prípade úspechu preložený ako aj spustený bez chýb.

Úloha 1.4

Ak ste vo svojom kóde vytvorili makrá na zadefinovanie vlastných farieb, presuňte ich do hlavičkového súboru colors.h.

Krok č. 2

Každá správna hra má vlastné menu, pomocou ktorého viete minimálne hru spustiť a ukončiť. V tomto kroku vytvoríte takéto menu aj pre svoju hru.

Úloha 2.1

Vytvorte jednoduché používateľské menu, ktoré sa zobrazí hneď po spustení hry a bude obsahovať minimálne položky: spustiť hru a ukončiť hru.

Význam jednotlivých položiek úvodného menu je nasledovný:

- spustiť hru - hra sa spustí od začiatku (level 0)
- ukončiť hru - hra sa korektne ukončí (uzatvorením všetkých otvorených súborov, uvoľnením celej obsadenej pamäte a pod; nestačí len zavolať funkciu exit())

Okrem uvedených položiek môžete do menu pridať napr. aj položku o autorovi.

Úloha 2.2

Overte správnosť svojej implementácie.

Krok č. 3

Aby sa aj v našom prípade jednalo o skutočnú hru, je potrebné vytvoriť systém, pomocou ktorého bude možné v hre po skončení jednej úrovne prejsť plynulo do ďalšej. V tomto kroku preto vytvoríte súbory levels.h a levels.c, v ktorých

sa tento systém bude nachádzať. Ak nebude uvedené ináč, budeme prioritne pracovať práve s týmito dvoma súbormi.

Úloha 3.1

Vytvorte nový hlavičkový súbor levels.h a k nemu prislúchajúci súbor so zdrojovým kódom levels.c.

Úloha 3.2

Vytvorte štruktúrovaný typ Level, ktorý bude reprezentovať jeden level hry a umiestnite ho do hlavičkového súboru levels.h.

Štruktúrovaný typ Level bude obsahovať nasledujúce položky:

- name - názov levelu
- description - (veselý) opis levelu, ktorý sa zobrazí pri jeho spúšťaní
- password - heslo pre vstup do daného levelu
- map - mapa samotného levelu v Sokoban formáte (http://sokobano.de/wiki/index.php?title=Level_format)
- next$_level - referencia na ďalší level$

Úloha 3.3

Vytvorte funkciu parse$_level(), ktorá rozparsuje reťazec odovzdaný funkcii ako parameter a vráti referenciu na adajový typ$

Táto funkcia teda:

- bude mať parameter typu char* a bude teda obsahovať reťazec znakov, a
- bude vracať referenciu na štruktúrovaný údajový typ Level.

Level je zadaný vo formáte, ktorý je opísaný v znení zadani. Pre otestovanie svojej funkcie môžete využiť pripravený súbor levels.dat, ktorý obsahuje 90 predpripravených levelov.

Úloha 3.4

Vytvorte funkciu load$_levels(), ktorá nám postupnosť levelov zo súboru a vráti referenciu na prvý z nich alebo vráti$ `null`, *ak*

Táto funkcia

- bude mať jeden parameter, ktorý bude špecifikovať cestu k súboru s levelmi uloženými na disku, a
- bude vracať referenciu na prvý level z postupnosti všetkých načítaných levelov alebo null, ak sa pri načítavaní vyskytne chyba.

Túto funkciu budete volať vo vašom programe len raz - na začiatku. Funkcia musí načítať úspešne všetky levely zo zadaného súboru, ktoré sú v ňom usporiadané na samostatných riadkoch (jeden riadok = jeden level, číslo riadku = číslo levelu).

Ak sa nepodarí z rozličných dôvodov načítať čo i len jeden z nich alebo súbor samotný nebude existovať, táto funkcia vráti hodnotu null, ktorá bude pre hru znamenať, že sa má ukončiť.

Ak sa funkcii nepodarí načítať jednotlivé levely správne a vráti null, nezabudnite na uvoľnenie pamäti a odstránenie z nej všetkých už načítaných levelov!

Pre otestovanie postupnosti levelov môžete s výhodou použiť súbor levels.dat, ktorý obsahuje 90 predpripravených levelov.

Samotnú postupnosť levelov reprezentujte ako jednosmerný spojkový zoznam.

Poznámka
Keďže budete potrebovať načítavať jednotlivé hodnoty dynamicky a nikdy si nebudete istí dĺžkou načítavaného reťazca, môžete využiť nasledovnú vlastnosť funkcií scanf() a fscanf() (zdroj: manuálová stránja funkcie scanf()):
The GNU C library supports a nonstandard extension that causes the library to dynamically allocate a string of sufficient size for input strings for the

```
1  char *p;
2  int n;
3
4  errno = 0;
5  n = scanf("%a[a-z]", &p);
6  if (n == 1) {
7      printf("read: %s\n", p);
8      free(p);
9  } else if (errno != 0) {
10     perror("scanf");
11 } else {
12     fprintf(stderr, "No matching characters\n");
13 }
```

As shown in the above example, it is only necessary to call free(3) if the scanf() call successfully read a string.
The a modifier is not available if the program is compiled with gcc -std=c99 or gcc -D$_{I}SOC99_{c}ode(unless_{G}NU_{c}odeisalsospecified), inwhichcasetheaisinterpretedasaspecifierforf$
$pointnumbers(seeabove)$.
Since version 2.7, glibc also provides the m modifier for the same purpose as the a modifier. The m modifier has the following advantages:

- It may also be applied to
- It avoids ambiguity with respect to the
- It is specified in the upcoming revision of the POSIX.1 standard.

Úloha 3.5

Vytvorte funkciu $\text{get}_{l}evel_{b}y_{p}assword(), ktorvrtiprslunlevelnazkladezadanhoheslaakoparametrafunkcie.$

Táto funkcia

- bude mať prvý parameter typu char*, ktorý bude reprezentovať heslo pre vstup do požadovaného levelu;
- druhý parameter typu Level*, ktorý bude referenciou na prvý level; a
- funkcia bude vracať referenciu na level, ku ktorému bude patriť zadané heslo. Ak však heslo nebude zadané, bude nesprávne, alebo parameter bude obsahovať NULL, funkcia vráti referenciu na prvý level.

Úloha 3.6

Vytvorte v menu hry novú položku Zadaj heslo, pomocou ktorej bude mať hráč možnosť zadať heslo pre vstup do príslušného levelu.

Ak hráč zadá správne heslo, otvorí sa mu prislúchajúci level, od ktorého môže následne hrať. Ak zadá nesprávne heslo, vráťte hráča do hlavného menu.

Úloha 3.7

Vytvorte funkciu $\text{levels}_{f}ree(), ktoruvonpamobsadenvetkmilevelmi.$

Táto funkcia bude mať jeden parameter, ktorý bude referenciou na prvý level z načítaného zoznamu levelov v pamäti. Funkciu budete volať pri ukončovaní hry.

Úloha 3.8

Overte správnosť svojej implementácie.

Overte správnosť implementácie každej vytvorenej funkcie.

Krok č. 4

V tomto kroku vytvoríte mechanizmus, pomocou ktorého overíte, či je level riešiteľný. Vytvoríte funkciu, ktorá na základe predaného reťazca, ktorý bude reprezentovať postupnosť krokov hráča v hre, a levelu, v ktorom sa má postupnosť overiť, zistí, či sa jedná alebo nejedná o riešenie levelu.

Úloha 4.1

Vytvorte funkciu $\text{check}_{s}olution_{f}or_{l}evel(), ktorover, izadanreazecpredstavujerieeniedanholevelu.$

Táto funkcia:

- bude mať prvý parameter typu Level*, ktorý bude referenciou na overovaný level, a

- druhý parameter bude typu char*, ktorý bude reprezentovať postupnosť krokov predstavujúcich riešenie.

Funkcia vráti hodnotu $True$, ak je daný level naozaj vyriešiteľný zadanou postupnosťou krokov, alebo $False$, ak uvedenou postupnosťou krokov vyriešiteľný nie je.

Reťazec reprezentujúci riešenie príslušného levelu implementujte podľa tohto opisu (http://sokobano.de/wiki/index.php?title=Level_format#Solution). Pri implementácii nie je potrebné rozlišovať veľkosť písmen.

Úloha 4.2

Overte správnosť svojho riešenia vytvorením reťazca prestavujúceho riešenie úvodných levelov v súbore levels.dat.

12.4 Doplňujúce úlohy

1. Podľa zadania sú levely uložené v čistej textovej forme, čo umožňuje jednoducho sa dostať ku ľubovoľnému heslu a teda umožňuje hráčovi hrať ľubovoľný level. Vytvorte preto funkcie char* code(char*) a char* decode(char*), z ktorých jedna bude vedieť zakódovať reťazec a druhá ho dekódovať. Na kódovanie môžete použiť ľubovoľný mechanizmus, resp. šifru (napr. môžete použiť jednoduchú substitučnú šifru známu pod názvom Cézarofa šifra).
2. Pridajte do hry kláves 'r', pomocou ktorého budete vedieť reštartovať práve rozohratý level.
3. Pridajte do menu položku *Redefine keys* (*Zmena kláves*), pomocou ktorej si bude môcť hráč nastaviť vlastné klávesy pre ovládanie hry.
4. Rozšírte proces zmeny kláves o možnosť zapnúť, resp. vypnúť tzv. *God mode*, kedy bude hráč môcť miesto hesla zadávať priamo čísla levelov, ktoré chce hrať. Samozrejme môžete do hry vložiť aj úplne iný *easter egg*, ktorý nemusí byť založený na zmene kláves, ale na postupnosti stlačených kláves počas hry.
5. Rozšírte štruktúrovaný typ Level o položku solution. Položka solution je reprezentovaná ako postupnosť hráčových ťahov na vyriešenie danej úrovne (formát (http://sokobano.de/wiki/index.php?title=Level_format#Solution)). Hráč sa pritom môže pohybovať len v smere hore (u), dole (d), vľavo (l) a vpravo (r). Príklad takéhoto reťazca je nasledujúci:

   DDrdrruLruLLDllU

   Smery s veľkými písmenami znamenajú, že Sokoban bude v danom smere pred sebou tlačiť bedňu. Riešenie však musí byť platné nehľadiac na veľkosť písmen.
   Túto položku môžete aktualizovať zakaždým, keď hráč level prejde a jeho riešenie je jednoduchšie (to znamená kratšie) ako to, ktoré je v aktuálnom leveli uvedené.

Obsah tejto položky môžete využiť ako demo pri hlavnom menu (buď ako samostatnú položku alebo sa demo spustí automaticky po uplynutí časového limitu). Počas behu dema sa samozrejme nezobrazí kompletné riešenie hry, ale len istý počet krokov z riešenia (max. polovica).

6. Vytvorte jednoduchý editor levelov, ktorý môže byť súčasťou samotnej hry alebo môže byť implementovaný ako samostatný program.
7. Vytvorte samostatnú (úvodnú obrazovku) pred spustením levelu. V tejto obrazovke nech sa zobrazí názov levelu, jeho opis a heslo do tohto levelu. Pre výpis týchto textov môžete vymyslieť vhodný efekt (napr. tzv. terminálový výpis, kedy sa každý znak vypíše s miernym spozdením).

12.5 Ďalšie zdroje

- Rudolf Pecinovský: Základy algoritmizace (http://publikace.pecinovsky.cz/Zaklady_algoritmizace.pdf)
- Pavel Herout: Učebnice jazyka C (1. díl) (http://www.martinus.sk/?uItem=74741)
- Formát reťazca pre riešenie úrovne Sokobana (http://sokobano.de/wiki/index.php?title=Level_format#Solution)
- Tutoriál k Makefile (http://www.opussoftware.com/tutorial/TutMakefile.htm)

Časť II

Informácie o predmete

13

14

15

16

17

Časť III

Návody

18

19

20

21

23

24