Základy algoritmizácie a programovania (2020)

Obsah

Rounds

Round 1

void pick_all()

Funkcia `void pick_all()` (1b)

Vytvorte funkciu `void pick_all()`, pomocou ktorej robot Karel pozbiera všetky značky na danej pozícii.

Return

Funkcia nevráti žiadnu hodnotu.

void find_south_east()

Funkcia `void find_south_east()` (2b)

Vytvorte funkciu `void find_south_east()`, ktorá premiestni robota Karla do pravého dolného rohu sveta

Return

Funkcia nevráti žiadnu hodnotu.

Počiatočná situácia

Na začiatku sa robot Karel nachádza na ľubovoľnej pozícii vo svete.

Koncová situácia

Na konci sa robot Karel nachádza na juhovýchode, teda v pravom dolnom rohu sveta. Je otočený smerom na západ.

bool facing_vertical()

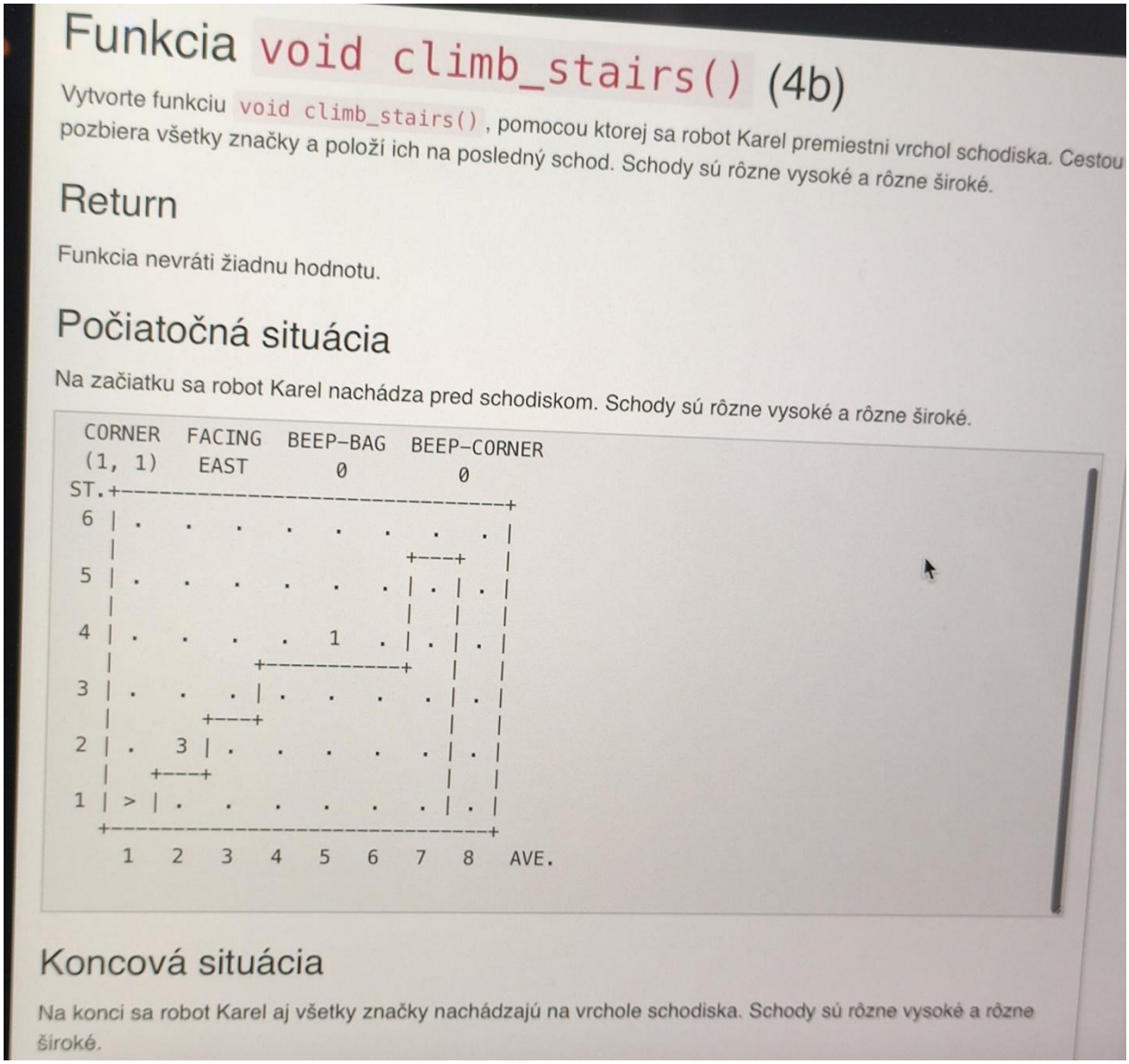
Funkcia `facing_vertical()` (2b)

Vytvorte funkciu `bool facing_vertical()`, ktorá zistí, či robot Karel má vertikálny smer (sever alebo juh).

Return

Funkcia vráti hodnotu `true`, ak robot smeruje na sever alebo juh, v opačnom prípade funkcia vráti hodnotu `false`.

void climb_stairs()



Funkcia `void climb_stairs()` (4b)

Vytvorte funkciu `void climb_stairs()`, pomocou ktorej sa robot Karel premiestni vrchol schodiska. Cestou pozbiera všetky značky a položí ich na posledný schod. Schody sú rôzne vysoké a rôzne široké.

Return

Funkcia nevráti žiadnu hodnotu.

Počiatočná situácia

Na začiatku sa robot Karel nachádza pred schodiskom. Schody sú rôzne vysoké a rôzne široké.

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

Koncová situácia

Na konci sa robot Karel aj všetky značky nachádzajú na vrchole schodiska. Schody sú rôzne vysoké a rôzne široké.

void olympics()

Funkcia `olympics()` (3b)

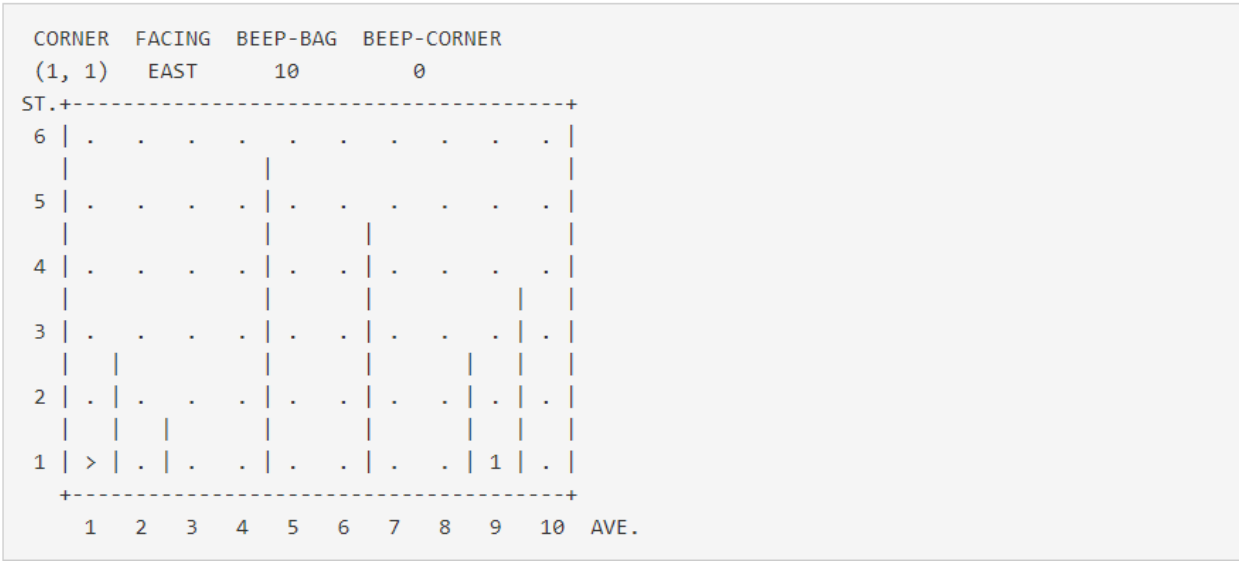
Vytvorte funkciu `void olympics()`, pomocou ktorej robot Karel preskočí ľubovoľne vysoké prekážky. Robot sa zastaví na značke.

Return

Funkcia nevráti žiadnu hodnotu.

Počiatočná situácia

Robot Karel stojí na začiatku prekážkovej trate.



void chessboard()

Funkcia `chessboard()` (4b)

Vytvorte funkciu `void chessboard()`, pomocou ktorej Karel dokáže vytvoriť šachovnicu ľubovoľnej veľkosti.

Čierne políčko je bez značky Biele políčko obsahuje značku. Karel bude začínať v ľavom dolnom rohu, kde je vždy políčko čierne.

Return

Funkcia nevráti žiadnu hodnotu.

Počiatočná situácia

Karel začína v ľavom dolnom rohu sveta (čierne políčko).

```
CORNER  FACING  BEEP-BAG  BEEP-CORNER
(5, 5)   NORTH     100        0
ST.+-------------------+
 5 | .   .   .   .   . |
   |                   |
 4 | .   .   .   .   . |
   |                   |
 3 | .   .   .   .   . |
   |                   |
 2 | .   .   .   .   . |
   |                   |
 1 | >   .   .   .   . |
   +-------------------+
     1   2   3   4   5   AVE.
```

void x_mas()

Funkcia `x_mas()` (4b)

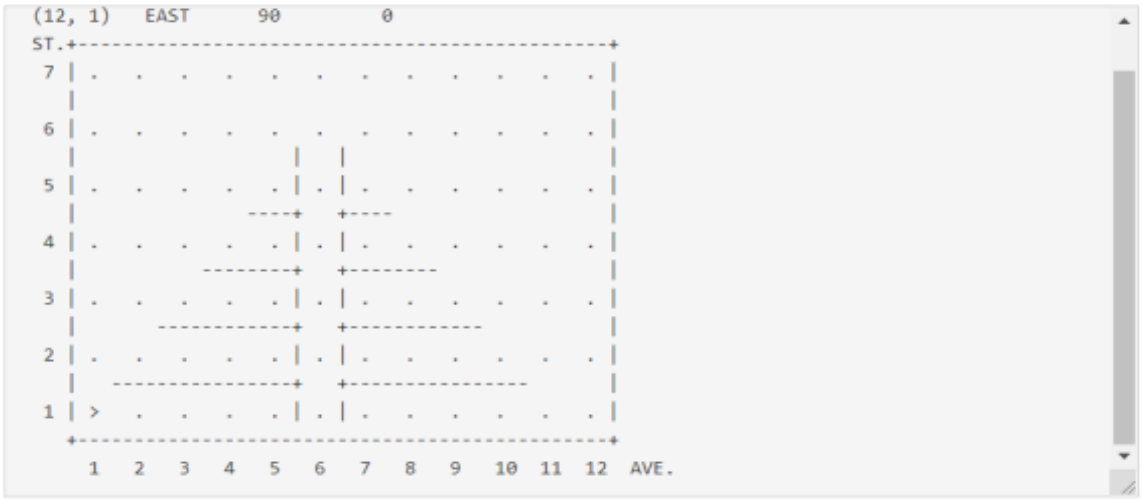
Vytvorte funkciu `void x_mas()`, pomocou ktorej pomôžete Karlovi ozdobiť vianočný strom, ak strom je už postavený. Ozdoby (značky) nech sa nachádzajú na kraji vetiev stromu a vrchol stromu nech zdobí dvojnásobná značka.

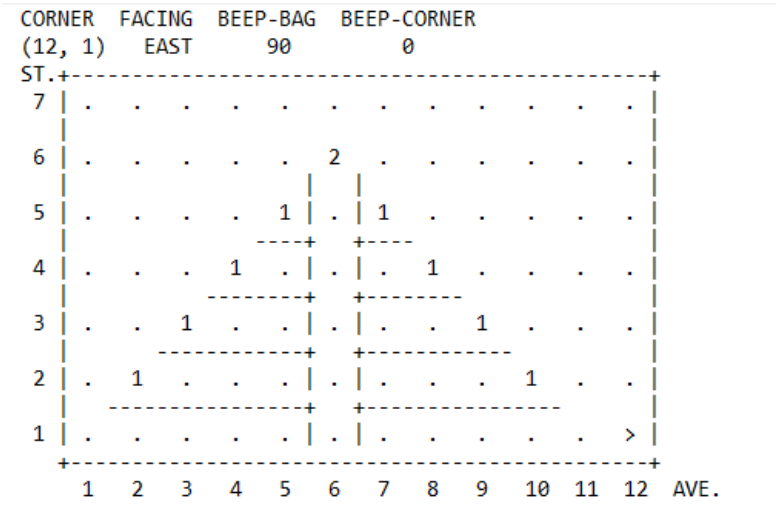
Return

Funkcia nevráti žiadnu hodnotu.

Počiatočná situácia

Karel začína v juhozápadnom rohu sveta.





void divide_even_beepers()

Funkcia `void divide_even_beepers()` (3b)

Vytvorte funkciu `void divide_even_beepers()`, pomocou ktorej robot Karel rozdelí kopu značiek na dve rovnaké polovice. Počet značiek je vždy párny.

Return

Funkcia nevráti žiadnu hodnotu.

Počiatočná situácia

Na začiatku sa robot Karel nachádza pred kopou značiek. Pred robotom sa nachádzajú aspoň dve pozície, na ktoré je možné rozdeliť kopu značiek. Batoh robota je prázdny.

```
 CORNER  FACING  BEEP-BAG  BEEP-CORNER
 (1, 2)   EAST       0         0
ST.+-----------+
 3 | .   .   . |
   |           |
 2 | >  10   . |
   |           |
 1 | .   .   . |
   +-----------+
     1   2   3   AVE.
```

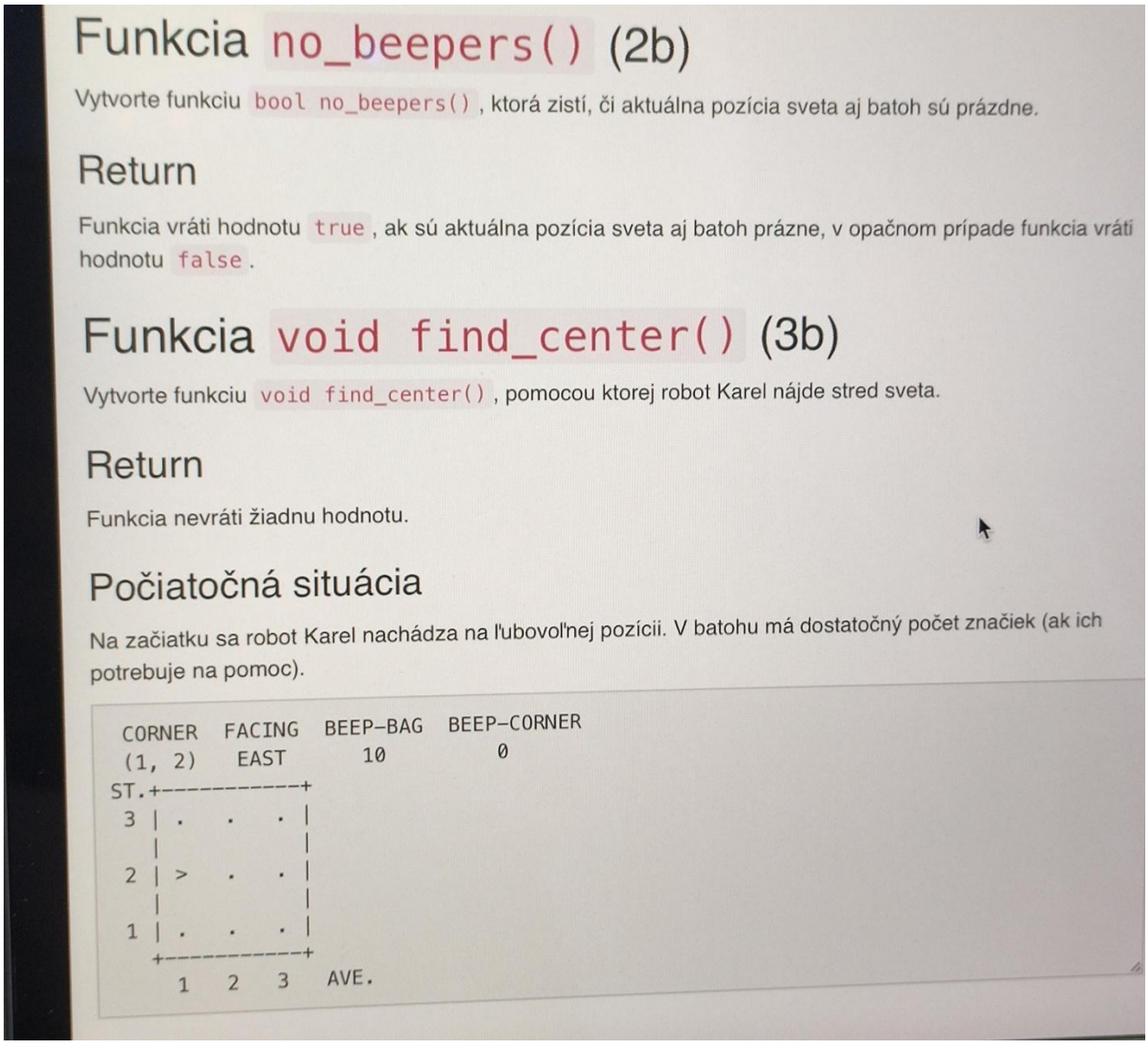
Koncová situácia

Na konci sa robot Karel nachádza na svojej pôvodnej pozícii, otočený smerom na východ. Kopa značiek je rozdelená na dve polovice. Prvá polovica značiek sa nachádza na svojej pôvodnej pozícii a druhá o jednu pozíciu napravo.

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

bool no_beepers()

void find_center()



Funkcia `no_beepers()` (2b)

Vytvorte funkciu `bool no_beepers()`, ktorá zistí, či aktuálna pozícia sveta aj batoh sú prázdne.

Return

Funkcia vráti hodnotu `true`, ak sú aktuálna pozícia sveta aj batoh prázne, v opačnom prípade funkcia vráti hodnotu `false`.

Funkcia `void find_center()` (3b)

Vytvorte funkciu `void find_center()`, pomocou ktorej robot Karel nájde stred sveta.

Return

Funkcia nevráti žiadnu hodnotu.

Počiatočná situácia

Na začiatku sa robot Karel nachádza na ľubovoľnej pozícii. V batohu má dostatočný počet značiek (ak ich potrebuje na pomoc).

```
 CORNER   FACING   BEEP-BAG   BEEP-CORNER
 (1, 2)    EAST       10          0
ST.+-----------+
 3 | .   .   . |
   |           |
 2 | >   .   . |
   |           |
 1 | .   .   . |
   +-----------+
     1   2   3   AVE.
```

void turn_to_south()

void mark_empty_line()

Funkcia `void turn_to_south()` (1b)

Vytvorte funkciu `void turn_to_south()`, pomocou ktorej sa robot Karel otočí na juh.

Return

Funkcia nevráti žiadnu hodnotu.

Funkcia `void mark_empty_line()` (2b)

Vytvorte funkciu `void mark_empty_line()`, ktorá označkuje jeden celý riadok alebo stĺpec značkami (podša smeru natočenia).

Return

Funkcia nevráti žiadnu hodnotu.

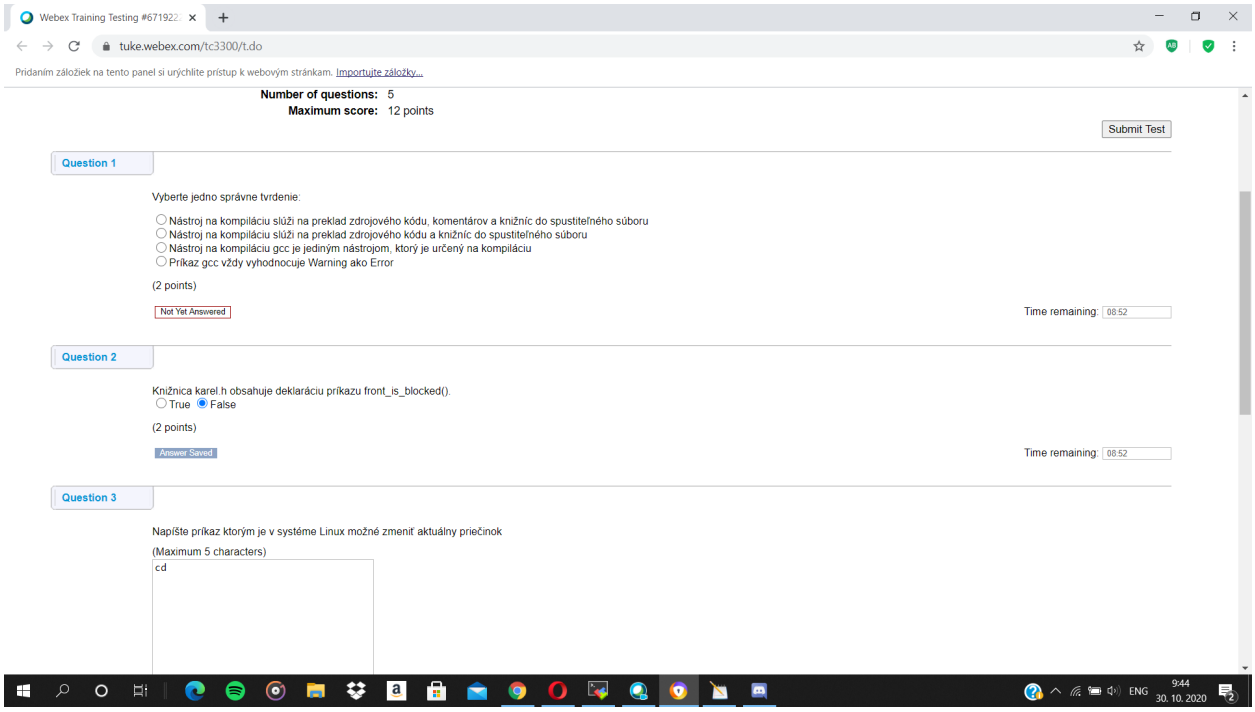
Počiatočná situácia

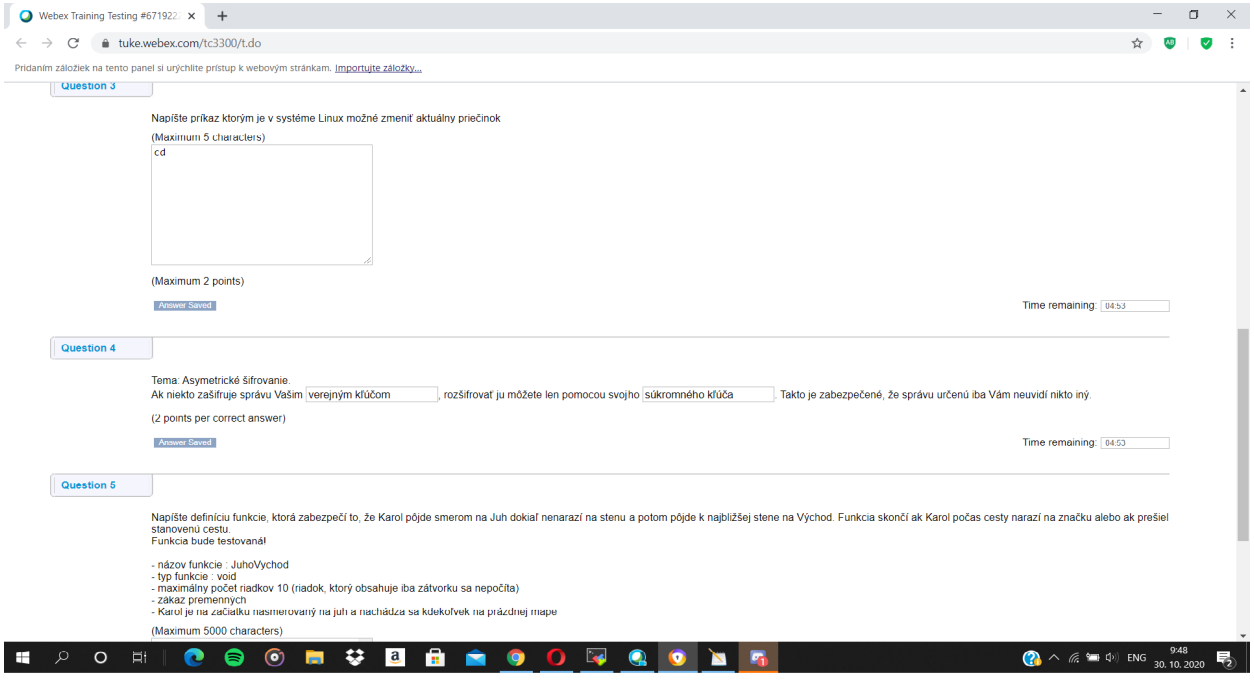
Na začiatku sa robot Karel nachádza na ľubovoľnej pozícii vo svete.

Koncová situácia

Podľa smeru robota bude na konci označkovaný celý riadok alebo celý stĺpec značkami. Na pozícii robota nezáleží.

Otázkový zápočet





ZAP - Round 1

Kód testu: 15

1. Robot Karel sa nachádza vo svete bez akýchkoľvek prekážok. Vytvorte funkciu s názvom find_home() s použitím knižnice superkarel.h, pomocou ktorej sa Karel dostane z akejkoľvek pozície do ľavého horného rohu sveta a otočí sa dole.

   (3 body)

2. Robor Karel sa nachádza vo svete bez prekážok v ľavom dolnom rohu, je otočený smerom nahor a vo vaku má jeden beeper. Vytvorte funkciu s názvom look_around(), pomocou ktorej Karel prejde svet popri jeho hraniciach v smere jeho natočenia a zastaví sa na pozícii, na ktorej začínal.

   (3 body)

3. Vytvorte funkciu s názvom easy_math(), na začiatku ktorej budú definované dve celé čísla. Ak budú tieto čísla rovnaké funkcia na monitor vypíše výsledok ich sčítania, ak nie funkcia vypíše výsledok odčítania nižšieho od vyššieho čísla.

   (3 body)

4. Vytvorte funkciu easy_physics(). Na začiatku funkcie budú definované štyri reálne čísla, z ktorých dve budú odpovedať dráham d1 a d2, ktoré teleso prešlo a ďalšie dve časom t1 a t2, za ktorý príslušnú dráhu prekonalo. Funkcia vypíše priemernú rýchlosť telesa.

   (3 body)

Round 2

int is_white(const char c)
void change_whites(char string [])

Funkcia `is_white()` (2b)

Vytvorte funkciu `int is_white(const char c)`, ktorá zistí, či je daný znak bielym znakom. Za biely znak považujeme znaky `' '`, `'\t'` a `'\n'`.

Input Params

- `const char c` - vstupný znak

Return

Funkcia vráti hodnotu `1`, ak vstupný znak je bielym znakom a hodnotu `0`, ak vstupný znak nie je bielym znakom.

Example (for main() only)

```
printf("%d %d\n", is_white('#'), is_white(' '));
// 0 1
```

Funkcia `change_whites()` (3b)

Vytvorte funkciu `void change_whites(char string[])`, ktorá zmení všetky biele znaky vstupného reťazca na bodky (znak `'.'`). Za biely znak sa považuje znak `' '` a znak `'\n'`.

Input Params

- `char string[]` - vstupný reťazec

Return

Funkcia nevráti žiadnu hodnotu, ale priamo zmení znaky reťazca. Avšak ak je vstupný reťazec `NULL`, funkcia nevykoná žiadnu operáciu (ukončí sa).

Example (for main() only)

```
char str[] = "Hello world!";
change_whites(str);
printf("%s\n", str);
// Hello.world!
```

int guess_eval(const int guess, const int my_number)
int leap_year(const int year)

Funkcia `guess_eval()` (1b)

Vytvorte funkciu `int guess_eval(const int guess, const int my_number)`, ktorá zistí, či hráč uhádol hádané číslo alebo či je hádané číslo väčšie/menšie.

Input Params

- `const int guess` - vstupné číslo, tip hráča
- `const int my_number` - vstupné číslo, hádané číslo

Return

Funkcia vráti hodnotu `1`, ak hráč uhádol hádané číslo. V opačnom prípade je potrebné vyhodnotiť, či je hádané číslo `my_number` väčšie ako tip hráča `guess` - vtedy funkcia vráti hodnotu `2`, alebo či je hádané číslo `my_number` menšie ako tip hráča `guess` - vtedy funkcia vráti hodnotu `0`

Example (for main() only)

```
printf("%d %d %d\n", guess_eval(34, 22), guess_eval(22, 34), guess_eval(34, 34));
// 0 2 1
```

Funkcia `leap_year()` (3b)

Vytvorte funkciu `int leap_year(const int year)`, ktorá zistí, či rok `year` je priestupným rokom (leap year). Pri určovaní priestupného roku uvažujte nasledovné pravidlá:

1. Každý 400. rok **je** priestupný
2. Každý 100. rok **nie je** priestupný
3. Každý 4. rok **je** priestupný
4. Každý rok **nie je** priestupný

V prípade konfliktu dvoch pravidiel vždy platí pravidlo umiestnené vyššie. Napr. rok 200 nie je priestupný, pretože hoci je deliteľný štyroma (3. pravidlo), je zároveň deliteľný stovkou (2. pravidlo). 2. pravidlo má teda prednosť pred 3. pravidlom.

Input Params

- `const int year` - vstupná hodnota, rok

Return

Funkcia vráti hodnotu `1`, ak zadaný rok je priestupný a hodnotu `0`, ak zadaný rok nie je priestupný. Avšak ak je rok menej ako `1` alebo ak je rok viac ako `4443`, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
printf("%d %d %d\n", leap_year(4000), leap_year(3000), leap_year(3004));
// 1 0 1
```

int count_positives(const int size, const int array[])

Funkcia `count_positives()` (3b)

Vytvorte funkciu `int count_positives(const int size, const int array[])`, ktorá zistí, koľko kladných čísel (väčších ako nula) obsahuje vstupné pole.

Input Params

- `const int size` - vstupná hodnota, veľkosť poľa
- `const int array[]` - vstupné pole

Return

Funkcia vráti hodnotu zodpovedajúcu počtu kladných čísel v poli (väčších ako nula). Avšak ak je vstupné pole `NULL`, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
const int array1[] = {1,2,0,3,4,0};
const int array2[] = {1,2,6,3,4,7};
const int array3[] = {-1,-2,0,-3,0,-2};
printf("%d %d %d\n", count_positives(6, array1), count_positives(6, array2), count_positives(6, array3));
// 4 6 0
```

int count_whites(const char string[])
int direction_correction(const int degree)

Funkcia `count_whites()` (3b)

Vytvorte funkciu `int count_whites(const char string[])`, ktorá spočíta všetky biele znaky v reťazci. Za biely znak považujeme znaky `' '`, `'\t'` a `'\n'`.

Input Params

- `const char string[]` - vstupný reťazec

Return

Funkcia vráti hodnotu zodpovedajúcu počtu bielych znakov v reťazci. Avšak ak je vstupné pole `NULL`, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
const char string[] = "Hello, how are you?";
printf("%d\n", count_whites(string));
// 3
```

Funkcia `direction_correction()` (3b)

Vytvorte funkciu `int direction_correction(const int degree)`, ktorá upraví stupeň otočenia robota Karla podľa nasledujúcich pravidiel::

1. Každých `90` stupňov predstavuje jeden smer. Základné stupne otočenia: `0`-východ, `90`-sever, `180`-západ, `270`-juh.
2. Ak je stupeň otočenia väčší ako základné stupne otočenia, je potrebné ho upraviť, napr. `360` na `0` alebo `450` na `90`.
3. Otočenie je neplatné, ak je stupeň otočenia `degree` záporný alebo ak nie je násobkom `90`.

Input Params

- `const int degree` - vstupné číslo, počet stupňov

Return

Funkcia vráti hodnotu zodpovedajúcu základnému stupňu otočenia robota Karla. Avšak ak je otočenie neplatné, funkcia vráti hodnotu `-1`.

Example

```
printf("%d %d %d\n", direction_correction(-90), direction_correction(540), direction_correction(180));
// -1 180 180
```

int all_positives(const int size, const int array[])

int last_positive(const int size, const int array[])

Funkcia `all_positives()` (3b)

Vytvorte funkciu `int all_positives(const int size, const int array[])`, ktorá zistí, či sú všetky hodnoty vstupného poľa kladné čísla (väčšie ako nula).

Input Params

- `const int size` - vstupná hodnota, veľkosť poľa
- `const int array[]` - vstupné pole

Return

Funkcia vráti hodnotu `1`, ak sú všetky hodnoty poľa kladné čísla (väčšie ako nula). V opačnom prípade vráti funkcia hodnotu `0`. Avšak ak je vstupné pole `NULL`, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
const int array1[] = {1,2,0,3,4,0};
const int array2[] = {1,2,6,3,4,7};
const int array3[] = {1,2,-1,3,4,-2};
printf("%d %d %d\n", all_positives(6, array1), all_positives(6, array2), all_positives(6, array3));
// 0 1 0
```

Funkcia `last_positive()` (3b)

Vytvorte funkciu `int last_positive(const int size, const int array[])`, ktorá nájde posledné kladné číslo v poli (väčšie ako `0`).

Input Params

- `const int size` - vstupná hodnota, veľkosť poľa
- `const int array[]` - vstupné pole

Return

Funkcia vráti hodnotu zodpovedajúcu poslednému kladnému číslu poľa (poslednému číslu väčšiemu ako `0`). Avšak ak je vstupné pole `NULL` alebo ak pole neobsahuje kladné čísla, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
const int array[] = {0,1,0};
const int array2[] = {-1,0,-6,-2};
printf("%d %d\n", last_positive(3,array), last_positive(4,array2));
// 1 -1
```

int binary_num(const int num)

Funkcia `binary_num()` (1b)

Vytvorte funkciu `int binary_num(const int num)`, ktorá zistí, či je vstupné číslo `0` alebo `1`. Vstupné číslo musí byť z rozsahu <-1000, 1000>.

Input Params

- `const int num` - vstupné číslo

Return

Funkcia vráti hodnotu `1`, ak je vstupné číslo `0` alebo `1`. V opačnom prípade funkcia vráti hodnotu `0`. Avšak ak je vstupné číslo mimo rozsah (je menšie ako `-1000` alebo väčšie ako `1000`), funkcia vráti hodnotu `-1`.

Example (for main() only)

```
printf("%d %d %d\n", binary_num(-1001), binary_num(3), binary_num(1));
// -1 0 1
```

void swap_sign(const int size, int array[])

Funkcia `swap_sign()` (3b)

Vytvorte funkciu `void swap_sign(const int size, int array[])`, ktorá otočí znamienka všetkých čísiel v poli.

Input Params

- `const int size` - vstupné číslo, veľkosť poľa
- `int array[]` - vstupné pole

Return

Funkcia nevráti žiadnu hodnotu, ale priamo upraví obsah poľa tak, aby sa záporné čísla stali kladnými a naopak, aby sa kladné čísla stali zápornými. Avšak ak je vstupné pole `NULL`, funkcia nevykoná žiadnu operáciu (ukončí sa).

Example (for main() only)

```
int array[] = {1,2,0,-3,4,0};
swap_sign(6, array);
for(int i = 0; i < 6; i++){
  printf("%d ", array[i]);
}
printf("\n");
// -1 -2 0 3 -4 0
```

int div_by_3(const int num)
int same_case(const char a, const char b)

Funkcia `div_by_3()` (1b)

Vytvorte funkciu `int div_by_3(const int num)`, ktorá zistí, či je vstupná hodnota číslo deliteľné číslom `3`.

Input Params

- `const int num` - vstupná hodnota

Return

Funkcia vráti hodnotu `1`, ak je vstupná hodnota `num` deliteľná číslom `3`, a hodnotu `0`, ak vstupná hodnota `num` nie je deliteľná číslom `3`.

Example (for main() only)

```
printf("%d %d %d\n", div_by_3(-3), div_by_3(6), div_by_3(8));
// 1 1 0
```

Funkcia `same_case()` (2b)

Vytvorte funkciu `int same_case(const char a, const char b)`, ktorá zistí, či sú obidva znaky malé resp. veľké písmená.

Input Params

- `const char a` - vstupný znak
- `const char b` - vstupný znak

Return

Funkcia vráti hodnotu `1`, ak sú obidva vstupné znaky malé písmená alebo ak sú obidva vstupné znaky veľké písmená. Funkcia vráti hodnotu `0`, ak je jeden znak veľké písmeno a druhý znak malé písmeno (alebo naopak). Avšak ak niektorý zo vstupných znakov nie je písmeno, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
printf("%d %d %d\n", same_case('a','f'), same_case('L','g'), same_case('#','P'));
// 1 0 -1
```

int direction_correction(const int degree)
int find_first_A(const char string[])
void string_to_upper(char string[])

Funkcia `direction_correction()` (3b)

Vytvorte funkciu `int direction_correction(const int degree)`, ktorá upraví stupeň otočenia robota Karla podľa nasledujúcich pravidiel::

1. Každých `90` stupňov predstavuje jeden smer. Základné stupne otočenia: `0`-východ, `90`-sever, `180`-západ, `270`-juh.
2. Ak je stupeň otočenia väčší ako základné stupne otočenia, je potrebné ho upraviť, napr. `360` na `0` alebo `450` na `90`.
3. Otočenie je neplatné, ak je stupeň otočenia `degree` záporný alebo ak nie je násobkom `90`.

Input Params

- `const int degree` - vstupné číslo, počet stupňov

Return

Funkcia vráti hodnotu zodpovedajúcu základnému stupňu otočenia robota Karla. Avšak ak je otočenie neplatné, funkcia vráti hodnotu `-1`.

Example

```
printf("%d %d %d\n", direction_correction(-90), direction_correction(540), direction_correction(180));
// -1 180 180
```

Funkcia `find_first_A()` (3b)

Vytvorte funkciu `int find_first_A(const char string[])`, ktorá nájde prvý znak `'A'` resp. `'a'` v reťazci. Na veľkosti písmen nezáleží.

Input Params

- `const char string[]` - vstupný reťazec

Return

Funkcia vráti hodnotu zodpovedajúcu indexu, na ktorom sa nachádza prvý znak `'A'` resp. `'a'` v reťazci (na veľkosti nezáleží). Avšak ak vstupný reťazec neobsahuje znak `'A'` ani `'a'` alebo ak je vstupný reťazec `NULL`, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
printf("%d\n", find_first_A("Tomorrow afternoon"));
// 9
```

Funkcia `string_to_upper()` (3b)

Vytvorte funkciu `void string_to_upper(char string[])`, ktorá všetky písmená v reťazci zmení na veľké písmená (lowercase to uppercase).

Input Params

- `char string[]` - vstupný reťazec

Return

Funkcia nevráti žiadnu hodnotu, ale priamo upraví obsah poľa `string` tak, aby všetky malé písmená sa zmenili na veľké písmená. Avšak ak je vstupné pole `NULL`, funkcia nevykoná žiadnu operáciu.

Round 3

int max_2d(const int size, int array[][size])
int vowels_count(const int rows, const int cols, char string[][cols])

Funkcia `max_2d()` (4b)

Vytvorte funkciu `int max_2d(const int size, int array[][size])`, ktorá nájde najväčšie číslo v dvojrozmernom štvorcovom poli.

Input Params

- `const int size` - vstupné číslo, veľkosť poľa
- `int array[][size]` - vstupné 2-rozmerné pole

Return

Funkcia vráti hodnotu zodpovedajúcu najväčšiemu číslu, ktoré sa nachádza v danom poli. Pole je vždy 2-rozmerné štvorcové (má rovnaký počet riadkov aj stĺpcov). Avšak ak je vstupné pole `NULL`, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
int array[2][2] = { {1,2}, {0,-3} };
printf("%d\n", max_2d(2, array));
// 2
```

Funkcia `vowels_count_2d()` (8b)

Vytvorte funkciu `int vowels_count_2d(const int rows, const int cols, char strings[][cols])`, ktorá spočíta samohlásky vo všetkých reťazcoch dvojrozmerného poľa.

Input Params

- `const int rows` - vstupná hodnota, počet riadkov poľa
- `const int cols` - vstupná hodnota, počet stĺpcov poľa
- `char strings[][cols]` - vstupné pole, na každom riadku obsahuje reťazec

Return

Funkcia vráti počet samohlások, ktoré sa nachádzajú vo všetkých reťazcoch dvojrozmerného poľa, ktorý je parametrom funkcie. Avšak ak je vstupné pole `NULL`, funkcia vráti hodnotu `-1`.

Example

```
char strings[3][50] = {"hello WORLD!", "aHOJ", "Ahoj"};
printf("%d\n", vowels_count_2d(3, 50, strings));
// 7
```

int is_in_array_2d(const int num, const int size, int array[][size])

Funkcia `is_in_array_2d()` (4b)

Vytvorte funkciu `int is_in_array_2d(const int num, const int size, int array[][size])`, ktorá zistí, či sa v dvojrozmernom poli nachádza konkrétne číslo.

Input Params

- `const int num` - vstupné číslo, hľadané číslo
- `const int size` - vstupné číslo, veľkosť poľa
- `int array[][size]` - vstupné 2-rozmerné pole

Return

Funkcia vráti hodnotu `1`, ak sa číslo `num` nachádza vo vstupnom poli `array`, a hodnotu `0`, ak sa tam nenachádza. Pole je vždy 2-rozmerné štvorcové (má rovnaký počet riadkov aj stĺpcov). Avšak ak je vstupné pole `NULL`, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
int array[2][2] = { {1,0}, {0,-3} };
printf("%d %d\n", is_in_array_2d(2, array), is_in_array_2d(-3, array));
// 0 1
```

Funkcia `largest_line()` (6b)

Vytvorte funkciu `int largest_line(const int size, int array[][size])`, ktorá nájde riadok s najväčším súčtom čísel v dvojrozmernom poli.

Input Params

- `const int size` - vstupné číslo, veľkosť poľa
- `int array[][size]` - vstupné 2-rozmerné pole

Return

Funkcia vráti hodnotu zodpovedajúcu indexu riadku (poľa), ktorý obsahuje najväčší súčet čísel. Ak pole obsahuje viac rovnakých súčtov, do úvah sa berie prvý z nich. Pole je vždy 2-rozmerné štvorcové (má rovnaký počet riadkov aj stĺpcov). Avšak ak je vstupné pole `NULL`, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
int array[2][2] = { {1,0}, {0,-3} };
printf("%d\n", largest_line(2, array));
// 0
```

Funkcia `swap_case_2d()` (6b)

Vytvorte funkciu `void swap_case_2d(const int rows, const int cols, char strings[][cols])`, ktorá v každom riadku 2d poľa (obsahuje reťazce) zmení všetky malé písmená reťazca na veľké a naopak všetky veľké písmená na malé.

Input Params

- `const int rows` - vstupná hodnota, počet riadkov poľa
- `const int cols` - vstupná hodnota, počet stĺpcov poľa
- `char strings[][cols]` - vstupné pole, na každom riadku obsahuje reťazec

Return

Funkcia nevráti žiadnu hodnotu, ale priamo upraví obsah reťazcov poľa. Avšak ak je vstupné pole `NULL`, funkcia nevykoná žiadnu operáciu (ukončí sa).

Funkcia `largest_col()` (6b)

Vytvorte funkciu `int largest_col(const int size, int array[][size])`, ktorá nájde STĹPEC s najväčším súčtom čísel v dvojrozmernom poli.

Input Params

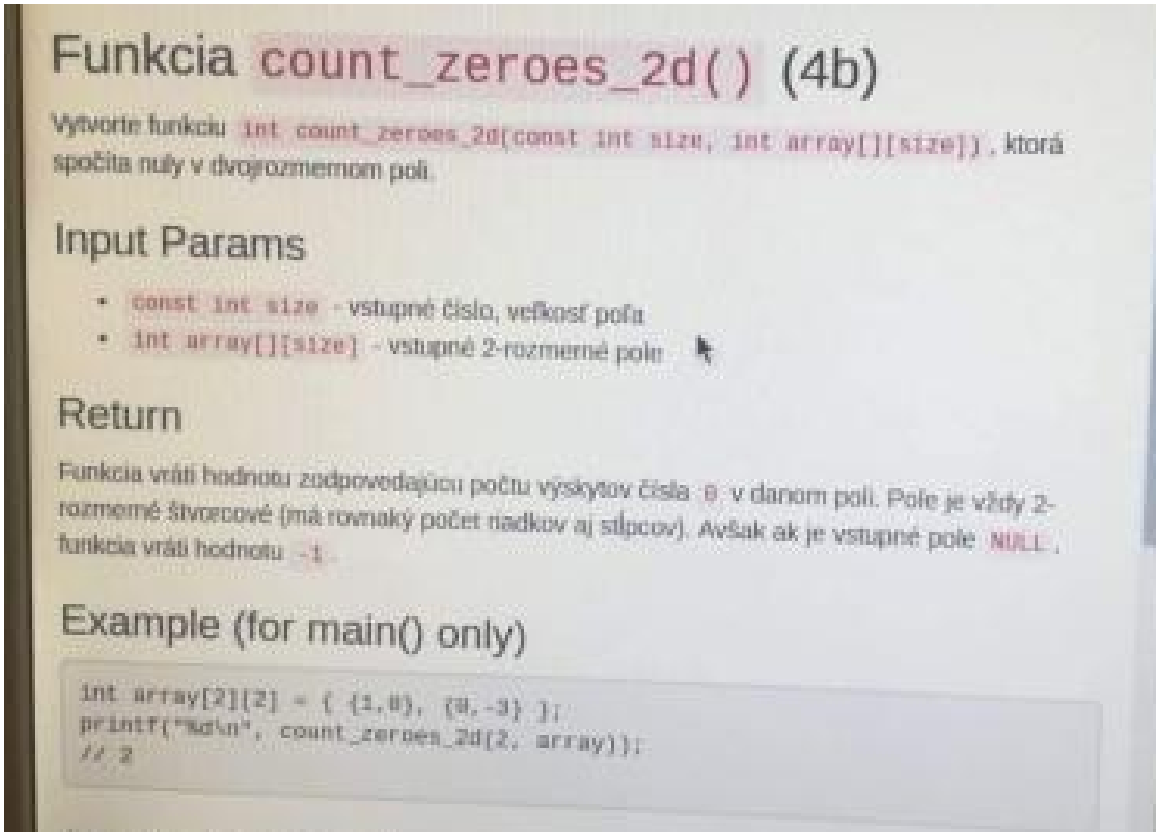
- `const int size` - vstupné číslo, veľkosť poľa
- `int array[][size]` - vstupné 2-rozmerné pole

Return

Funkcia vráti hodnotu zodpovedajúcu INDEXU STĹPCA, ktorý obsahuje najväčší súčet čísel. Ak pole obsahuje viac rovnakých súčtov, do úvahy sa berie prvý z nich. Pole je vždy 2-rozmerné štvorcové (má rovnaký počet riadkov aj stĺpcov). Avšak ak je vstupné pole `NULL`, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
int array[2][2] = { {1,0}, {0,-3} };
printf("%d\n", largest_col(2, array));
// 0
```



Funkcia `count_zeroes_2d()` (4b)

Vytvorte funkciu `int count_zeroes_2d(const int size, int array[][size])`, ktorá spočíta nuly v dvojrozmernom poli.

Input Params

- `const int size` - vstupné číslo, veľkosť poľa
- `int array[][size]` - vstupné 2-rozmerné pole

Return

Funkcia vráti hodnotu zodpovedajúcu počtu výskytov čísla `0` v danom poli. Pole je vždy 2-rozmerné štvorcové (má rovnaký počet riadkov aj stĺpcov). Avšak ak je vstupné pole `NULL`, funkcia vráti hodnotu `-1`.

Example (for main() only)

```
int array[2][2] = { {1,0}, {0,-3} };
printf("%d\n", count_zeroes_2d(2, array));
// 2
```

Funkcia `swap_sign_2d()` (4b)

Vytvorte funkciu `void swap_sign_2d(const int size, int array[][size])`, ktorá otočí znamienka všetkých čísiel v dvojrozmernom poli.

Input Params

- `const int size` - vstupné číslo, veľkosť poľa
- `int array[][size]` - vstupné 2-rozmerné pole

Return

Funkcia nevráti žiadnu hodnotu, ale priamo upraví obsah poľa tak, aby sa záporné čísla stali kladnými a naopak, aby sa kladné čísla stali zápornými. Avšak ak je vstupné pole `NULL`, funkcia nevykoná žiadnu operáciu (ukončí sa).