STU FIIT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Základy procedurálneho programovania 2

FIIT STU, Mlynská dolina

Aula Minor, pondelok 9:00

letný semester

2016/2017

Ideme podľa plánu

dátum	prednáška	8:00	9:00	cvičenie	obsah
20.3.	6		Čítanie kódu, Hľadanie chýb v kóde	6	Projekt 1: odovzdanie
27.3.	7	Test 3	Riešenie testu 3, Spájané zoznamy	7	Projekt 2 **Tezeus a Minotaurus**
3.4.	8		Tezeus, Bitové operácie	8	
10.4.	9	Test 4	Riešenie testu 4, Rekurzia, Minotaurus	9	
17.4.	Veľká noc			X	
24.4.	10		Ďalšie prvky jazyka C	10	Projekt 2: odovzdanie, konzultovanie
1.5.	Sviatok			11	
8.5.	Sviatok			12	
9.5.	11		Opakovanie	X	
15.5.	12	Predtermín?		X	

Opakovanie ... Úloha A

- Daná je funkcia f nad reálnymi číslami (reprezentovanými ako double) a hodnota x, napíš hlavičku funkcie **deriv**, ktorá dostane f a x ako vstupné argumenty, vypočíta hodnotu derivácie funkcie f v bode x a vráti ju ako návratovú hodnotu. Nepoužívajte globálne premenné.

- Riešenie?
- **double deriv(double (*f)(double), double x);**

Opakovanie ... Úloha B

B (max. 2b): Daný je zdrojový kód (vľavo) a zistite, ktoré upozornenia a chyby obsahuje a do tabuľky (vpravo) uveďte riadky zdrojového kódu, v ktorých sa nachádzajú. Otáznik ??? zakrýva konkrétny názov.

```
 1  #include <stdlib.h>
 2  int g[10][10], c[10];
 3
 4  int main()
 5  {
 6    int i, k;
 7    while (scanf("%d", &i) > 0)
 8    {
 9      g[i/10][i%10]++;
10      c[i/10]++;
11    }
12    for (i < 0; i = 10; i++)
13      if (c[i])
14      {
15        printf("%d | ", i);
16        for (j = 0; j < 10 j++)
17          while (g[i][k] = 0)
18          {
19            printf('%d', j);
20            g[i][j];
21          }
22        printf('\n');
23      }
24    return 0;
25  }
```

Chyby alebo upozornenia	Riadky
error: '???' undeclared (first use in this function)	
error: expected ';' before '???'	
warning: passing argument 1 of '???' makes pointer from integer without a cast	
warning: statement with no effect	
warning: suggest parentheses around assignment used as truth value	
warning: multi-character character constant	
warning: implicit declaration of function '???'	

Opakovanie ... Úloha B (riešenie)

```
1  #include <stdlib.h>
2  int g[10][10], c[10];
3
4  int main()
5  {
6    int i, k;
7    while (scanf("%d", &i) > 0)
8    {
9      g[i/10][i%10]++;
10     c[i/10]++;
11   }
12   for (i < 0; i = 10; i++)
13     if (c[i])
14     {
15       printf("%d | ", i);
16       for (j = 0; j < 10 j++)
17         while (g[i][k] = 0)
18         {
19           printf('%d', j);
20           g[i][j];
21         }
22       printf('\n');
23     }
24   return 0;
25 }
```

Chyby alebo upozornenia	Riadky
error: '???' undeclared (first use in this function)	16
error: expected ';' before '???'	16
warning: passing argument 1 of '???' makes pointer from integer without a cast	19, 22
warning: statement with no effect	12
warning: suggest parentheses around assignment used as truth value	12, 17
warning: multi-character character constant	19
warning: implicit declaration of function '???'	7, 15

STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Základy procedurálneho programovania 2

Makrá, preprocesor

3. 4. 2017

letný semester
2016/2017

Činnosť preprocesora

- spracováva zdrojový text **PRED** kompilátorom
- zamieňa text, napr. identifikátory konštánt za číselné hodnoty
- vypustí zo zdrojového textu všetky komentáre
- prevádza podmienený preklad
- nekontroluje syntaktickú správnosť programu

- riadok, ktorý má spracovávať preprocesor sa začína znakom #

Konštrukcie pre preprocesor

- definovanie makra

  #define meno_makra text

- zrušenie definície makra

  #undef meno_makra

- podmienený preklad v závislosti na konštante **konst**

  #if konst

  #elif #else #endif

Konštrukcie pre preprocesor

- vloženie textu zo špecifikovaného súbora zo systémového adresára

  #include <filename>

- vloženie textu zo špecifikovaného súbora v adresári používateľa

  #include "filename"

- výpis chybových správ vo fáze predspracovania

  #error text

Konštrukcie pre preprocesor

- podmienený preklad v závislosti od toho, či je makro definované, alebo nedefinované

  #ifdef meno_makra

  #elif #else #endif

- podmienený preklad v závislosti od toho, či je makro nedefinované, alebo definované

  #ifndef meno_makra

  #elif #else #endif

Konštanty - makrá bez parametrov

- symbolické konštanty
- používajú sa často (zbavujú program "magických čísel")
- väčšinou definované na začiatku modulu
- platnosť konštánt je do konca modulu
- náhrada konštanty hodnotou - rozvoj (expanzia) makra

Pravidlá pre písanie konštánt

- mená konštánt - veľkými písmenami
- meno konštanty je od hodnoty oddelené apsoň jednou medzerou
- za hodnotou by mal byť vysvetľujúci komentár
- nové konštanty môžu využívať skôr definované konštanty
- ak je hodnota konštanty dlhšia ako riadok, musí byť na konci riadku znak **** (nie je súčasťou makra)

Príklady definovania konštánt

```
#define MAX 1000
#define PI 3.14
#define DVE_PI (2 * PI)
#define MOD %
#define AND &&
#define MENO_SUBORU "list.txt"
#define DLHA_KONSTANTA Toto je dlha konstanta, \
            ktora sa nezmesti do jednoho riadku.
```

- za hodnotou nie je ;
- medzi menom konštanty a jej hodnotou nie je =

Príklad požitia konštanty: výpočet obsahu kruhu

```
#include <stdio.h>
#define PI 3.14

int main(){
    double r;

    printf("Zadajte polomer: ");
    scanf("%lf" &r);
    printf("Obvod kruhu s polomerom %f je %f\n",
                r, 2 * r * PI);
   return 0;
}
```

Príklad použitia konštanty: malé písmená zmení na veľké

```
#include <stdio.h>
#define POSUN ('a' - 'A')
#define EOLN '\n'
#define PRED_MALE  '*'

int main() {
   int c;

   while((c = getchar()) ! = EOLN) {
      if (c >= 'a' && c <= 'z') {
         putchar(PRED_MALE);
         putchar(c - POSUN);
      }
      else
         putchar(c);
   }
   return 0;
}
```

ak je symbolickou konštantou výraz, vhodné je uzavrieť ho do zátvoriek

malé písmeno zmení na veľké a pred neho vypíše '*', inak vypíše načítaný znak

Kedy sa nerozvinie makro

- makro sa nerozvinie, ak je uzatvorené v úvodzovkách

```
#define MENO   "Katka"

...
printf("Volam sa MENO");

printf("Volam sa %s", MENO);
```

vypíše sa:
Volam sa MENO

vypíše sa:
Volam sa Katka

Prekrývanie definícií

- nová definícia prekrýva starú, pokiaľ je rovnaká (to ani nemá zmysel)
- ak nie je rovnaká:
  - zrušiť starú definíciu: **#undef meno_makra**
  - definovať meno_makra

```
#define POCET 10
#undef POCET
#define POCET 20
```

Makro ako skrytá časť programu

```
#define ERROR { printf("Chyba v datach.\n"); }
```

- pri použití nie je makro ukončené bodkočiarkou:

```
if (x == 0)
    ERROR
else
    y = y / x;
```

Makrá s parametrami

- krátka a často používaná funkcia vykonávajúca jednoduchý výpočet
  - problém s efektivitou (prenášanie parametrov a úschova návratovej hodnoty je časovo náročnejšia ako výpočet)
  - preto namiesto funkcie - makro (to sa pri preprocesingu rozvinie)

- je potrebné sa rozhodnúť medzi
  - funkcia: kratší ale pomalší program
  - makro: rýchlejší ale dlhší program

Makrá s parametrami

- nazývajú sa vkladané funkcie - rozvitie makra znamená, že sa meno makra nahradí jeho telom

definícia makra

```
#define je_velke(c) ((c) >= 'A' && (c) <= 'Z')
```

- zátvorka, v ktorej sú argumenty funkcie - hneď za názvom makra (bez medzery)

v zdrojovom súbore

```
ch = je_velke(ch) ? ch + ('a' - 'A') : ch;
```

rozvinie sa

```
ch = ((ch) >= 'A' && (ch) <= 'Z') ? ch + ('a'-'A') : ch;
```

Makrá s parametrami

- telo makra - uzavrieť do zátvoriek, inak môžu nastať chyby, napr.

Nesprávne:

```
#define sq(x) x * x
...
sq(f + g); f + g * f + g;
```

po rozvinutí makra

Správne:

```
#define sq(x) ((x) * (x))
...
sq(f + g); ((f + g) * (f + g));
```

po rozvinutí makra

Preddefinované makrá

- **`getchar()`** a **`putchar()`** (v **`stdio.h`**)

```
#define getchar() getc(stdin)
#define putchar(c) putc(c, stdout)
```

- makrá v **`ctype.h`** - makrá na určenie typu znaku
  - **`isalnum`** - vráti 1, ak je znak číslica alebo malé písmeno
  - **`isalpha`** - vráti 1, ak je znak malé alebo veľké písmeno
  - **`isascii`** - vráti 1, ak je znak ASCII znak (0 až 127)
  - **`iscntrl`** - vráti 1, ak je znak Ctrl znak (1 až 26)
  - ...

Preddefinované makrá

- makrá v **`ctype.h`** - makrá na konverziu znaku
  - **`tolower`** - konverzia na malé písmeno
  - **`toupper`** - konverzia na veľké písmeno
  - **`toascii`** - prevod na ASCII - len najnižších 7 bitov je významných

Vkladanie súborov

- vkladanie systémových súborov **< >**
- vkladanie súborov v aktuálnom adresári **" "**

```
#include <stdio.h>
#include <ctype.h>
#include "KONSTANTY.H"
```

Podmienený preklad

- u väčších programov
  - ladiace časti - napr. pomocné výpisy
- program
  - trvalá časť
  - voliteľná časť (napr. pri ladení, alebo ak je argumentom programu nejaký prepínač)

Riadenie prekladu hodnotou konštantného výrazu

```
#if konstantny_vyraz
    cast_1
#else
    cast_2
#endif
```

ak je hodnota konštantného výrazu nenulová, vykoná sa časť 1, inak časť 2

```
#if 0
    cast programu, co
    ma byt vynechana
#endif
```

ak pri testovaní nechcete prekladať časť programu, namiesto `/* */` (problém by robili vhniezdené komentáre)

Riadenie prekladu hodnotou konštantného makra

```
#define PCAT 1

#if PCAT
    #include <conio.h>
#else
    #include <stdio.h>
#endif
```

- ak je program závislý na konkrétnom počítači
- ak na PC/AT - definujeme PCAT na 1, inak na 0

Riadenie prekladu definíciou makra

```
#define PCAT

#ifdef PCAT
    #include <conio.h>
#else
    #include <stdio.h>
#endif
```

- ak je program závislý na konkrétnom počítači
- ak na PC/AT - definujeme PCAT (bez hodnoty),
- stačí, že je konštanta definovaná

```
#ifndef PCAT
```

- ak nie je definovaná konštanta

```
#undef PCAT
```

- zrušenie definície makra

Operátory defined, #elif a #error

- **#ifdef**, alebo **#ifndef** zisťujú existenciu len jedného symbolu, čo neumožňuje kombinovať viaceré

- ak treba kombinovať viaceré podmienky:

  #if defined TEST **#if !defined TEST**

- **#elif** - má význam else-if

- **#error** - umožňuje výpis chybových správ (v priebehu preprocesingu - nespustí sa kompilácia)

Operátory defined, #elif a #error - príklad

```
#if defined(ZAKLADNY)    &&    defined(DEBUG)
    #define VERZIA_LADENIA  1
#elif defined(STREDNY)    &&    defined(DEBUG)
    #define VERZIA_LADENIA 2
#elif !define(DEBUG)
    #error Ladiacu verziu nie je mozne pripravit!
#else
    #define VERZIA_LADENIA  3
#endif
```

STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Základy procedurálneho programovania 2

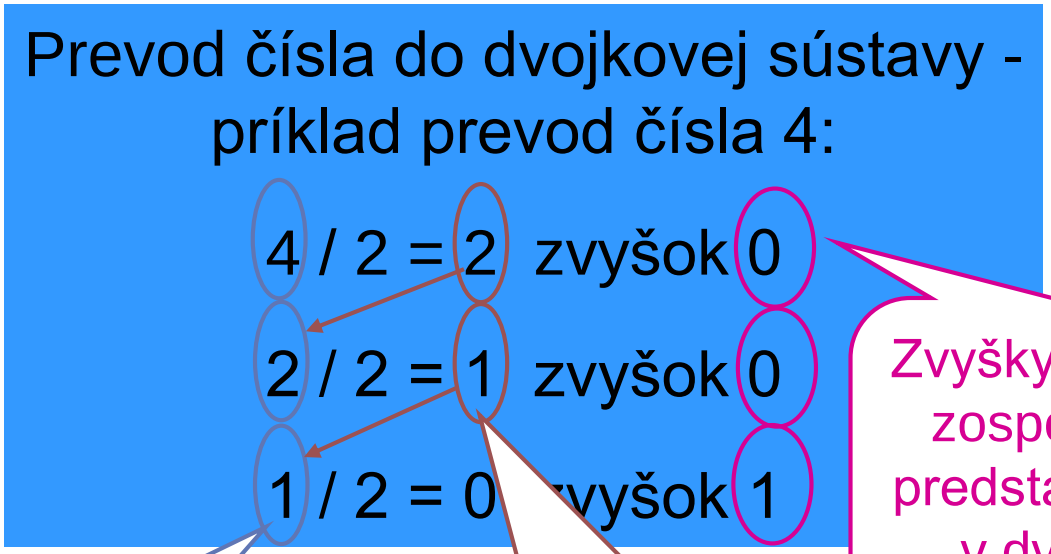
Bitové operácie

3. 4. 2017

letný semester 2016/2017

Práca s bitmi

- práca s reprezentáciou čísla v dvojkovej sústave
- Príklady:
  - 1: 001
  - 2: 010
  - 3: 011
  - 4: 100



Zvyšky prečítané zospodu hore predstavujú číslo v dvojkovej sústave

Prevod čísla do dvojkovej sústavy (delenie dvomi)

Výsledok sa použije ako delenec v nasledujúcej časti prevodu

Operácie s jednotlivými bitmi

- operátory:
  - **&** - bitový súčin (AND)
  - **|** - bitový súčet (OR)
  - **^** - bitový exkluzívny súčet (XOR)
  - **<<** - posun doľava
  - **>>** - posun doprava
  - **~** - jednotkový doplnok (negácia bit po bite)
- argumenty nemôžu byť **float**, **double** ani **long double**

Bitový súčin

- **i**-ty bit výsledku **x & y** bude 1 vtedy, ak **i**-ty bit **x** aj **i**-ty bit **y** sú 1, inak 0 (*AND* po bitoch)

```
#define je_neparne(x) (1 & (unsigned) (x))
```

```
  0000 0000 0000 0001
& xxxx xxxx xxxx xxx1
-------------------
  0000 0000 0000 0001
```

x	y	x&y
0	0	0
0	1	0
1	0	0
1	1	1

- ak chceme premennú typu int použiť ako ASCII znak, teda potrebujeme najmenších 7 bitov

```
0000 0000 0111 1111 = 0x7F
```

```
c = c & 0x7F;
```

```
c &= 0x7F;
```

Rozdiel medzi bitovým a logickým súčinom

```
unsigned int i = 1, j = 2, k, l;
k = i && j;
l = i & j;
```

- **k**: 1, pretože 1 a 2 sú kladné číla, teda majú logickú hodnotu *true (pravda)* a **&&** je logický súčin

- **l**: 0, pretože 1 = 0000 0001
  2 = 0000 0010
  a **&** je bitový súčin

Bitový súčet

- **i**-ty bit výsledku **x | y** bude 1 vtedy, ak **i**-ty bit **x** alebo **i**-ty bit **y** sú 1, inak 0 (*OR* po bitoch)
- používa sa na nastavenie niektorých bitov na jednotku, pričom nechá ostatné bity nezmenené

```
#define na_neparne(x) (1 | (unsigned) (x))
```

```
  0000 0000 0000 0001
| xxxx xxxx xxxx xxx1
-------------------
  xxxx xxxx xxxx xxx1
```

x	y	x|y
0	0	0
0	1	1
1	0	1
1	1	1

makro vráti nepárne číslo nezmenené a párne zväčší o 1

Bitový exkluzívny súčet

- **i**-ty bit výsledku **x ^ y** bude 1 vtedy, ak sa **i**-ty bit **x** nerovná **i**-temu bitu **y**, inak 0 (*XOR* po bitoch)

```
if (x ^ y) /* cisla sú rozdielne */
```

x	y	x^y
0	0	0
0	1	1
1	0	1
1	1	0

Bitový posun doľava

- **x << n** posunie bity v **x** o **n** pozícií doľava
- bity zľava sa strácajú bity zprava sú dopĺňané nulami

```
x = x << 1;
```

na rýchle násobenie dvomi

```
x = 0001 1011 0010 0101  = 6949
x << 1 = 0011 0110 0100 1010  = 13898 = 2 * 6949
```

```
x = x << 3;
```

vynásobesnie $2^3 = 8$

Bitový posun doprava

- **x >> n** posunie bity v **x** o **n** pozícií doprava
- bity sprava sa strácajú bity zľava sú dopĺňané nulami

```
x = x >> 1;
```

na rýchle celočíselné delenie dvomi

```
x = 0011 0110 0100    = 13898
x >> 1 = 0001 1011 0010  = 6949 = 13898 / 2
         0101
```

```
x = x >> 3;
```

celočíselné delenie $2^3 = 8$

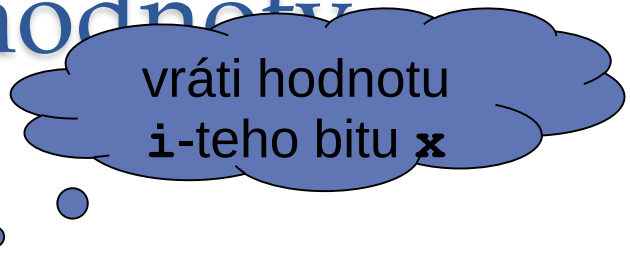
Príklad: delenie a násobenie

- bitové posuny sú rýchlejšie ako násobenie a delenie násobkami dvojky

```
i = j * 80;
i = (j << 6) + (j << 4);
```

80 = 64 + 16
rýchlejšie

Príklad: zistenie hodnoty konkrétneho bitu




```
#define ERROR -1
#define CLEAR 1
#define BIT_V_CHAR 8

int bit(unsigned x, unsigned i)
{
  if (i >= sizeof(x) * BIT_V_CHAR)
    return (ERROR);
  else
    return ((x >> i) & CLEAR);
}
```

```
  0011 0010 0101 0010
  0001 1001 0010 1001
  0000 1100 1001 0100
  0000 0110 0100 1010
  0000 0011 0010 0101
  0000 0001 1001 0010
  0000 0000 1100 1001

  0000 0000 1100 1001
& 0000 0000 0000 0001
  ---------------------
  0000 0000 0000 0001
```

Negácia po bitoch

- jednotkový doplnok **~x**
- prevráti nulové bity na jednotkové a naopak
- Použitie napr. ak sa chceme vyhnúť na počítači závislej dĺžke celého čísla:

```
x &= 0xFFF0;
```

nastavenie posledných 4 bitov na nulu - len ak platí

```
sizeof(int) == 2
```

```
x &= ~0xF;
```

nastavenie posledných 4 bitov na nulu - platí pre všade

Príklad

Zistenie dĺžkyky typu **int** v bitoch

```
#include <stdio.h>

int dlzka_int(){
   unsigned int x, i = 0;

   x = ~0;      /* negácia 0 -> same 1 */

   while ((x >> 1) != 0)
      i++;
   return (++i);
}

int main(){
   printf("Dlzka typu int je %d bitov\n", dlzka_int());
   return 0;
}
```

Práca so skupinou bitov

- stavová premenná **stav** - definuje práva na prístup k súboru

```
#define READ 0x8
#define WRITE 0x10
#define DELETE 0x20

unsigned int stav;

stav |= READ | WRITE | DELETE;
stav |= READ | WRITE;
stav  &= ~(READ | WRITE | DELETE);
stav &= ~READ;
if ( ! (stav & (WRITE | DELETE)))
   ...
```

→ **READ:** 2^3 = 0000 1000
→ **WRITE:** 2^4 = 0001 0000
→ **DELETE:** 2^5 = 0010 0000

nastaví 3., 4. a 5. bit na 1

nastaví 3., 4. bit na 1

nastaví 3., 4. a 5. bit na 0

nastaví 3. bit na 0

ak 3. a 4. bit sú nulové

Bitové pole

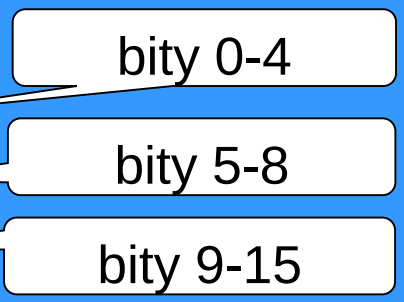
- štruktúra, ktorej veľkosť je obmedzená veľkosťou typu **int**
- najmenšia dĺžka položky je 1 bit
- definuje podobne ako štruktúra, ale každá položka bitového poľa je určená menom a dĺžkou v bitoch
- môže byť **signed** aj **unsigned** (preto vždy uviesť)
- oblasti použitia:
  - uloženie viac celých čísel v jednom (šetrenie pamäte)
  - pre prístup k jednotlivým bitom (často)

Príklad bitového poľa

- uloženie dátumu do jednotho **int**-u:
  - deň - najmenších 5 bitov,
  - mesiac - ďalšie 4 bity,
  - rok - zvyšných 7 bitov (max. 127, preto rok - 1980)

```
typedef struct {
    unsigned den    : 5;
    unsigned mesiac : 4;
    unsigned rok    : 7;
} DATUM;

DATUM dnes, zajtra;
dnes.den = 29;
dnes.mesiac = 11;
dnes.rok = 2012 - 1980;
zajtra.den = dnes.den + 1;
```



Príklad bitového poľa

Dátum ako bitové pole aj
hexadecimálne číslo (union)

```
#include <stdio.h>

typedef struct {
  unsigned den    : 5;   /* bity 0 -  4 */
  unsigned mesiac : 4;   /* bity 5 -  8 */
  unsigned rok    : 7;   /* bity 9 - 15 */
} DATUM;

typedef union {
  DATUM          datum;
  unsigned int   cislo;
} BITY;
```

```c
#include <stdio.h>

typedef struct {
  unsigned den     : 5,
  unsigned mesiac  : 4,
  unsigned rok     : 7,
} DATUM;

typedef union {
  DATUM          datum;
  unsigned int   cislo;
} BITY;

int main(void)
{
  BITY dnes;
  int d, m, r;

  printf("Zadaj dnesny datum [dd mm rrrr]: ");
  scanf("%d %d %d", &d, &m, &r);
  dnes.datum.den = d;
  dnes.datum.mesiac = m;
  dnes.datum.rok = r - 1980;

  printf("datum: %2d.%2d.%4d - cislo: %X hexa\n",
         dnes.datum.den, dnes.datum.mesiac,
         dnes.datum.rok + 1980, dnes.cislo);
  return 0;
}
```

STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Základy procedurálneho programovania 2

Projekt 2: Tezeus a Minotaurus

27.3.2017

letný semester 2016/2017

Projekt 2: Tezeus a Minotaurus

- Tezeus chce v labyrinte nájsť a zneškodniť Minotaura
- Vstup:
  - Získal mapu labyrintu s určenými význačnými bodmi
- Tezeus:
  - Nájsť susedné význačné body
  - Kreslenie obrázku – mapa labyrintu (s oblasťami)

Projekt 2: Tezeus

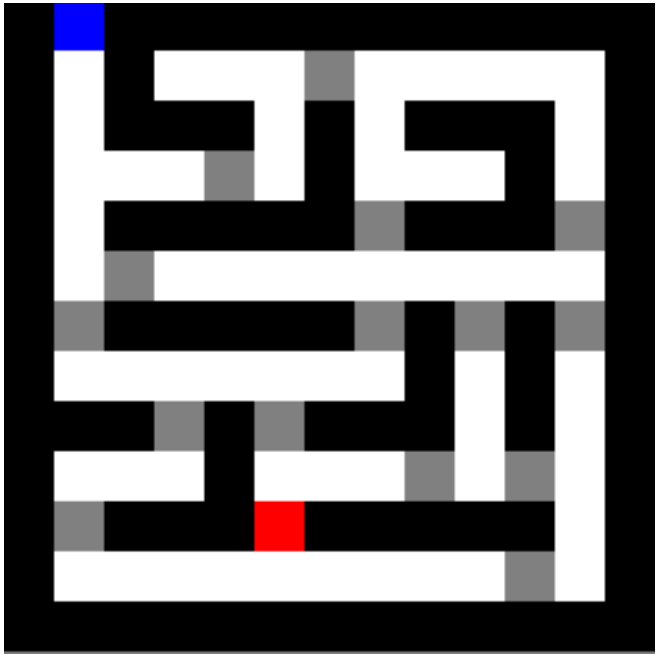
- Mapa labyrintu s určenými význačnými bodmi

- Tezeus musí: **nájsť susedné význačné body**
- Napr.:
  - T → A
  - T → B
  - T → Q
  - A → C
  - ...

```
#T###########
#.#...R.....#
#.###.#.###.#
#...Q.#...#.#
#.#####C###F#
#.A.........#
#B######E#K#L#
#.......#.#.#
###D#H###.#.#
#...#...J.P.#
#G###X#####.#
#.........N.#
#############
```

Projekt 2: Tezeus

- Kreslenie obrázku (BMP) – Mapa labyrintu
  - Steny (čierne), chodby (biele)
  - Tezeus (modrý), Minotaurus(červený)



Projekt 2: Tezeus

- Kreslenie obrázku (BMP) – Mapa labyrintu
  - Steny (čierne), chodby (biele)
  - Tezeus (modrý), Minotaurus(červený)
  - Súvislé oblasti medzi význačnými bodmi (rozličné farby)

Projekt 2: Tezeus

- Mapa labyrintu s určenými význačnými bodmi

- Tezeus musí: **nájsť susedné význačné body**
- Napr.:
  - T → A
  - T → B
  - T → Q
  - A → C
  - ...

```
#T###########
#.#...R.....#
#.###.#.###.#
#...Q.#...#.#
#.#####C###F#
#.A.........#
#B#####E#K#L#
#.......#.#.#
###D#H###.#.#
#...#...J.P.#
#G###X#####.#
#.........N.#
#############
```

Projekt 2: Minotaurus

- Tezeus sa chce vrátiť naspäť:
  - Na lístoček si píše význačné body, ktorými už prešiel
  - Lístoček pojme obmedzený počet položiek (najviac K)
  - Do ktorých oblastí labyrintu sa môže Tezeus dostať, keď sa mu na lístoček zmestí najviac K význačných bodov?

- Minotaurus:
  - Minotaurus sa chce schovať niekde ďalej v labyrinte
  - Kreslenie obrázku – do ktorých oblastí sa Tezeus môže dostať v závislosti od hodnoty K

Nabudúce...

dátum	prednáška	8:00	9:00	cvičenie	obsah
20.3.	6		Čítanie kódu, Hľadanie chýb v kóde	6	Projekt 1: odovzdanie
27.3.	7	Test 3	Riešenie testu 3, Spájané zoznamy	7	Projekt 2 **Tezeus a Minotaurus**
3.4.	8		Tezeus, Bitové operácie	8	
10.4.	9	Test 4	Riešenie testu 4, Rekurzia, Minotaurus	9	
17.4.	Veľká noc			X	
24.4.	10		Ďalšie prvky jazyka C	10	Projekt 2: odovzdanie, konzultovanie
1.5.	Sviatok			11	
8.5.	Sviatok			12	
9.5.	11		Opakovanie	X	
15.5.	12	Predtermín?		X	