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

Ako sa pracuje na tomto predmete?

- **Prednášky** (pondelok 9:00):
  Jozef Tvarožek
  zapájať sa, pýtať sa, počúvať, ... písomné testy (20%)
- **Cvičenia** (streda 12:00, 14:00, 16:00, štvrtok 11.00):
  Marián Potočný
  riešenie malých úloh, projekty (40%)
  www.turing.sk/fiit/zpp2

- Povinné minimum zo semestra 30 bodov
- **Skúška:**
  písomný test (40%) (povinná účasť)

Organizácia po týždňoch (1. – 6.)

dátum	prednáška	8:00	9:00	cvičenie	obsah
13.2.	1		Opakovanie	1	Projekt 1 **Snehulienka**
20.2.	2		Algoritmické obchodovanie	2	
27.2.	3	Test 1	Riešenie testu 1, Snehulienka	3	
6.3.	4		Snehulienka (pokr.)	4	
13.3.	5	Test 2	Riešenie testu 2, Smerníky	5	Projekt 1: odovzdanie, konzultovanie
20.3.	6		Čítanie kódu, Hľadanie chýb v kóde	6	

Organizácia po týždňoch (7. – 12.)

dátum	prednáška	8:00	9:00	cvičenie	obsah
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. (ut)	11		Opakovanie	X	
15.5.	12	Predtermín?		X	

Čo by ste už mali vedieť (ZPrPr)

- Premenné, zložené (polia, štruktúry), smerníky
- Výrazy (aritmetický, logický)
- Príkazy (vetvetnie, cyklus)
- Funkcie
- Reťazce
- Údaje v pamäti
- Alokácia a dealokácia
- Načítavania (scanf)
- Výpis (printf)
- Kreslenie ASCII
- Bludisko

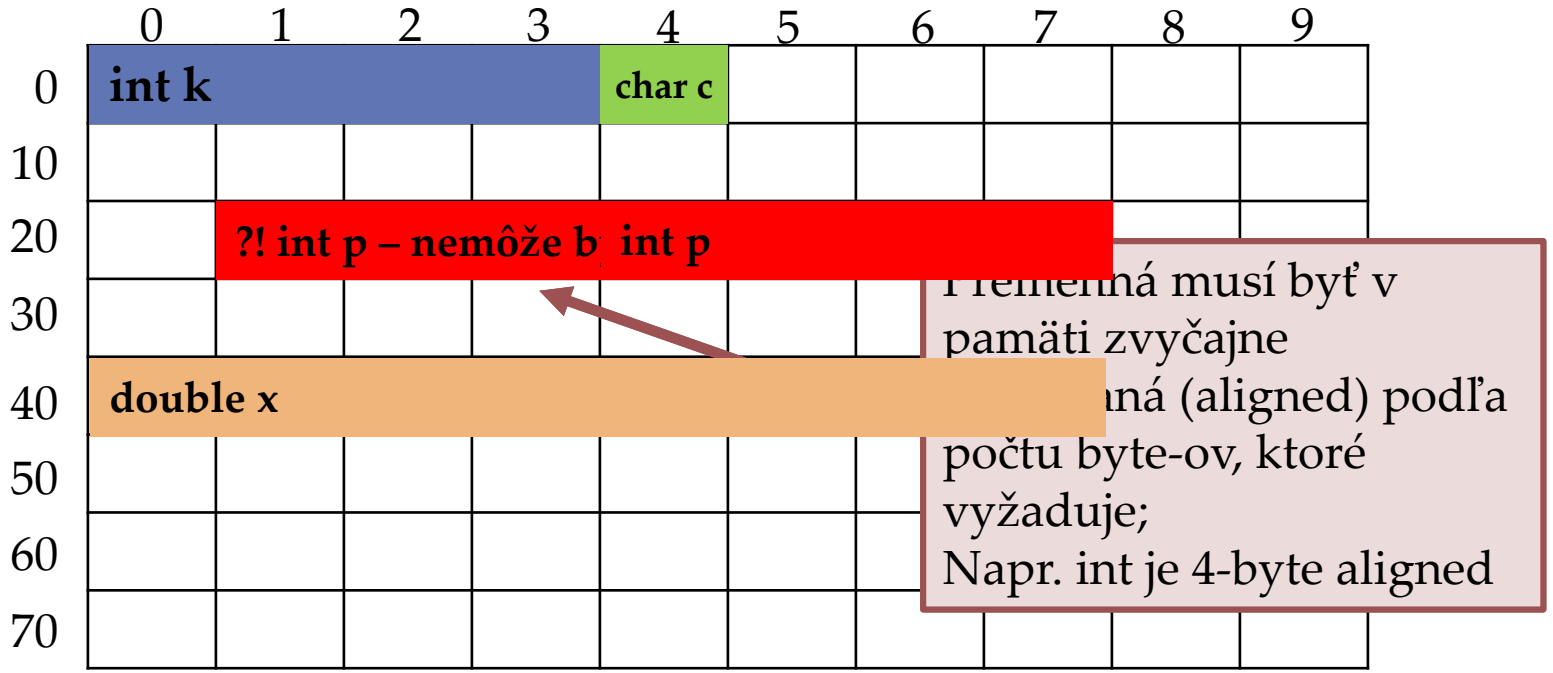
Opakovanie – Premenná

- Rozličné dáta, ktoré ukladáme do pamäti zaberajú v pamäti rozlične veľa miesta
- **Adresa pamäte** je poradie (číslo) byte-u od začiatku pamäte
- **Identifikátory** odkazuju na entity (premenné, funkcie) v programe
  - Názvy premenných, funkcií, ...
- **Premenná** je pomenovaný priestor v pamäti
  - previazanie identifikátora s pamaťou
- **Priradenie** naplní hodnotu do premennej (do pamäti vyhradenej pre premennú) **vek** ← **24**

Opakovanie – Premenná

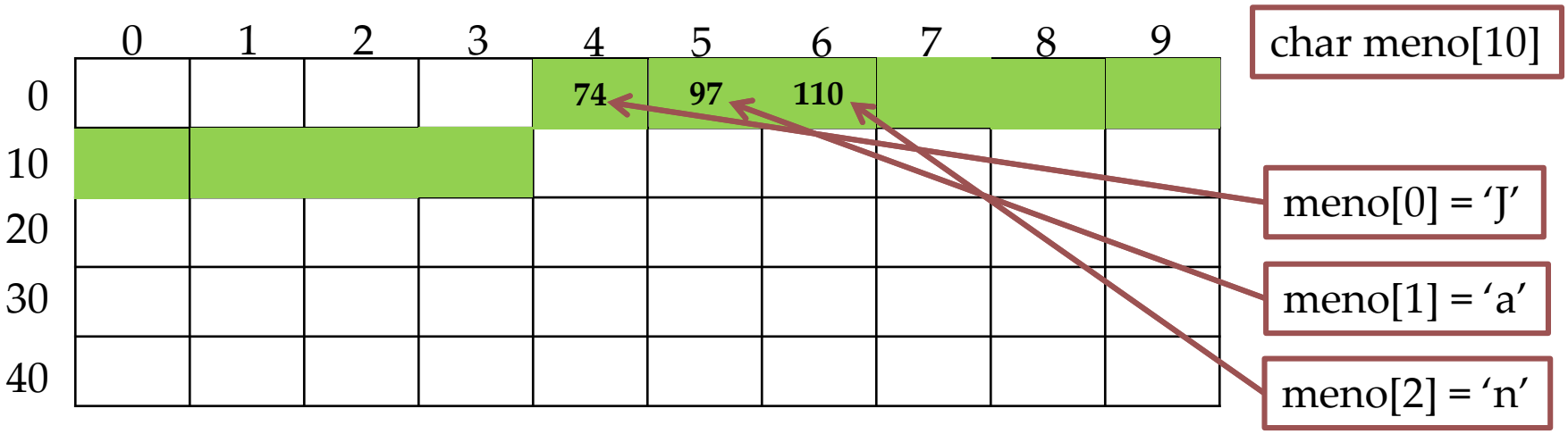
- **Premenná** je pomenovaný priestor v pamäti
  - previazanie identifikátora s pamaťou
- Začiatok (prvý byte) v pamäti, kde sú dáta pre premennú vyhradené, nazývame **adresa premennej**
  - **adresa je (obyčajné) číslo** – poradie prvého byte od začiatku (vyhradenej) pamäte
- Adresa sa označuje symbolom ampersand (&)
- Premenná x – **adresa x je &x**
- **Priradenie cez adresu** (px = &x)
  ***px = 42** je to isté ako **x = 42**

Opakovanie – Premenná v pamäti



- Ako si do pamäte uložím znak? (napr. 'x')
- Ako si do pamäte uložím meno? (napr. „Bratislava“)

Opakovanie – Reťazec znakov v pamäti



- Ako zistím dĺžku (počet znakov hodnoty) reťazca?
- Pre premennú meno je v pamäti vyhradených 10 znakov, ak je hodnota reťazca „Jan“, tak má len 3 znaky
- Riešenie: za posledným znakom hodnoty reťazca je číslo 0
- Reťazec „Jan“ je teda:

74	97	110	0

- Reťazec „0+2=5“ je:

48	43	50	61	53	0

Opakovanie – Reťazec v jazyku C

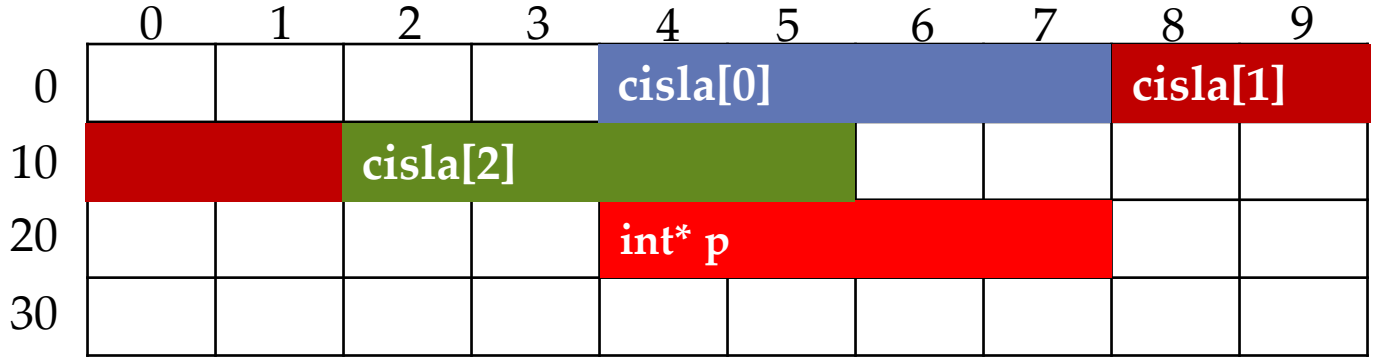
- Reťazec je pole znakov (char)
  napr. char meno[5]
- Hodnota reťazca sú znaky až po ukončovací kód 0
- **Výpis reťazca** na obrazovku
  printf a formátovací reťazec %s:

```
char meno[5];
meno[0] = 'J';
meno[1] = 'a';
meno[2] = 'n';
meno[3] = 0;
printf("%s", meno); // vypise Jan
```

- Čo keď v reťazci nie je ukončenie kódom 0?
  - Vypisuje sa pamäť, až kým sa niekde ďalej nenarazí na kód 0
    Napr.: Jan#$TR#RF#E%9^#$@#T#TG#@T%

Opakovanie – Zložené dátové typy – pole

- Viac rovnakých dátových prvkov za sebou v pamäti
- Môžeme k prvkom pristupovať výpočtom presného miesta od začiatku.
- Napr. **int cisla[3]** …



- Adresa premennej cisla? ... Prvý byte v pamäti – 4
- Adresa tretieho čísla? ... &cisla[2] ... 12
- Nech: p=cisla; *(p+1) = 40; … kam sa naplní 40?

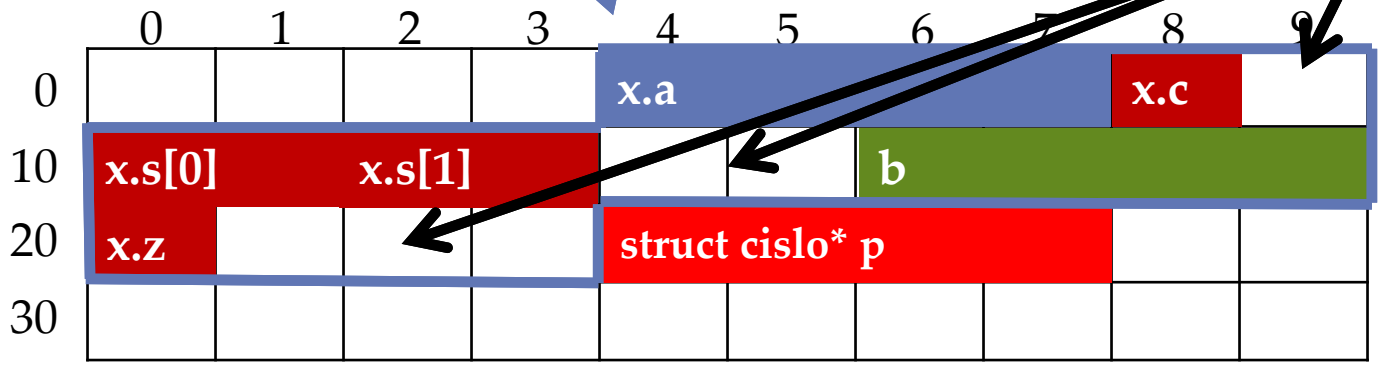
Opakovanie – Zložené dátové typy – štruktúra

- **struct cislo** {
  int a;
  char c;
  short s[2];
  int b;
  char z;
  }

struct cislo x;

Modrý okraj: Celková veľkosť vyhradenej pamäte pre jednu štruktúru typu **struct cislo** je 20 bytes (celá musí byť zarovnaná na 4 bytes)

Nevyužité miesta, kvôli zarovnaniu



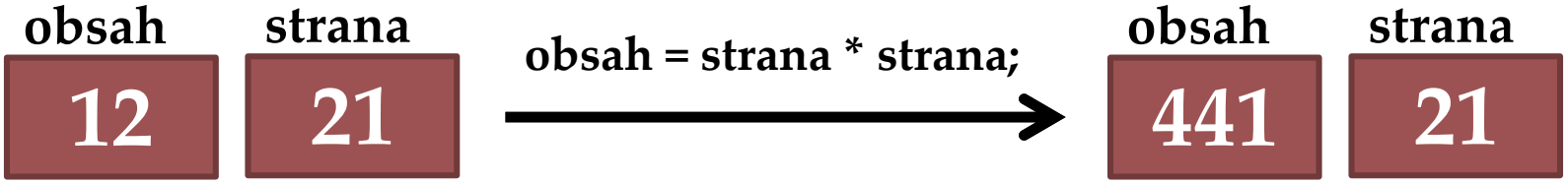
- Adresa premennej x? ... Prvý byte v pamäti – 4
- Adresa druhého čísla v poli x.s[1]? ... &x.s[1] ... 12
- Nech: p=&x; (*p).c = 'w'; Pre prístup cez smerník na štruktúru môžeme priamo použiť **šípku** (p->c = 'w')

Opakovanie – Výrazy, príkazy

- **Výraz (aritmetický)**
  - Vyhodnotenie výrazov
  - Operátory (+, -, =, ...) a operandy (premenné, konštanty)
- **Logický výraz**
  - AND, OR, NOT, ...
  - Skrátené vyhodnocovanie
- **Vetvenie (if / else)**
- **Cyklus (for / while)**
  - **for** (premenné; podmienka; krok)
    { blok; }

Opakovanie – Ukážka vyhodnotenia výrazu

- **obsah ← strana × strana**
- Rovnica v jazyku C **obsah = strana * strana;** sú **inštrukcie**, ktoré vykonajú:
  1) prečítajú hodnotu v pamäti vyhradenej pre premennú *strana*
  2) **vyhodnotia aritmetický výraz** strana * strana procesor hodnoty vynásobí (operácia *)
  3) výslednú hodnotu súčinu zapíše (uloží) do pamäte vyhradenej pre premennú *obsah*



Opakovanie – Prerušenie cyklu

- **Break** – prerušenie vykonávania cyklu
- **Continue** – ukončiť vykonávanie aktuálnej iterácie, a pokračovať na skok, podmienku (a možno ďalšiu iteráciu)

```
for (int i = 0; i < 10; i++)
{
  if (i % 2 == 0)
     continue;      ak je i párne nepokračuj, ale choď na ďalšie
  for (int j = 0; j < 10; j++)
  {
    if (i == j)
      break;        ak je i rovné j skonči vykonávanie tohto cyklu
    PRINT (10*i + j)
  }
  PRINT newline
}
```

i

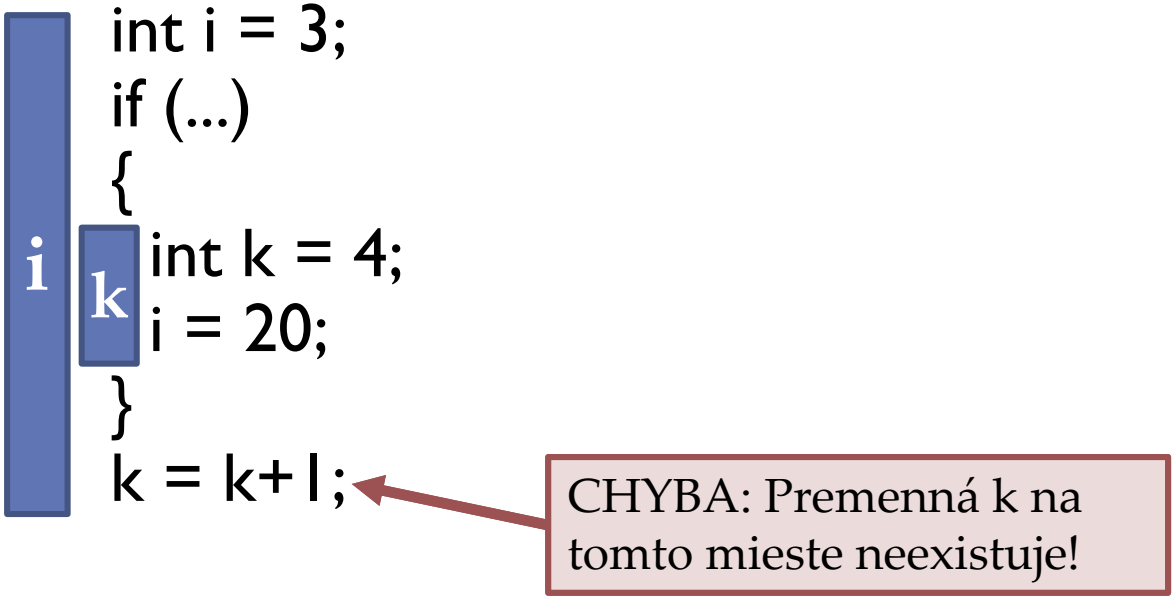
j

- Čo to vypíše?

~~0 1 2 3 4 5 6 7 8 9~~
10 ~~11~~ ~~12 13 14 15 16 17 18 19~~
~~20 21 22 23 24 25 26 27 28 29~~
30 31 32 ~~33~~ ~~34 35 36 37 38 39~~
~~40 41 42 43 44 45 46 47 48 49~~
50 51 52 53 54 ~~55~~ ~~56 57 58 59~~
…

Opakovanie – Funkcie

- **Rozsah platnosti** nám hovorí, kde v programe môžeme identifikátor použiť, a s ktorou pamäťou je identifikátor previazaný
- **Blok** – časť kódu, ktorá je zoskupená spolu, typicky **ohraničená zloženými zátvorkami { }**
- Príklad (rozsah platnosti premenných vyznačený stĺpcami):



Opakovanie – Funkcie

- **Funkcia** je pomenovaná časť programu, ktorá vykonáva určitú úlohu
  - Funkcia je tiež uložená v pamäti podobne ako premenné, pričom jej dáta sú samotné inštrukcie, ktoré funkcia vykonáva)
- Funkcia je definovaná ako:

```
typ nazovFunkcie(argumenty)
{
  blok;      -- telo funkcie
}
```

- Argumenty je zoznam pomenovaných dátových typov, ktoré nadobudnú platnosť v rozsahu bežiacej funkcie ako premenné.
- **return** je príkaz, ktorý ukončí funkciu, a ako návratovú hodnotu vráti príslušnú hodnotu.

Opakovanie – Funkcie

- Návratovú hodnotu funkcie potom môžeme použiť tam, kde funkciu voláme (tzv. **volanie funkcie**).
  int x=10, y=20, obvod;
  obvod = **obvod_obdlznika**(x, y);
  PRINT obvod
- Volanie funkcie vytvorí nový rozsah platnosti pre parametre funkcie
  - Ak v príklade vyššie x, a y upravíme vo funkcií obvod_obdlznika, tak sa zmeny neprejavia vo volajúcej funkcií
  - **Ak chceme vykonať zmeny v hodnotách argumentov funkcie, musíme do funkcie poslať adresy premenných** – aby funkcia mala počas volania potrebné informácie, ktoré údaje zmeniť.

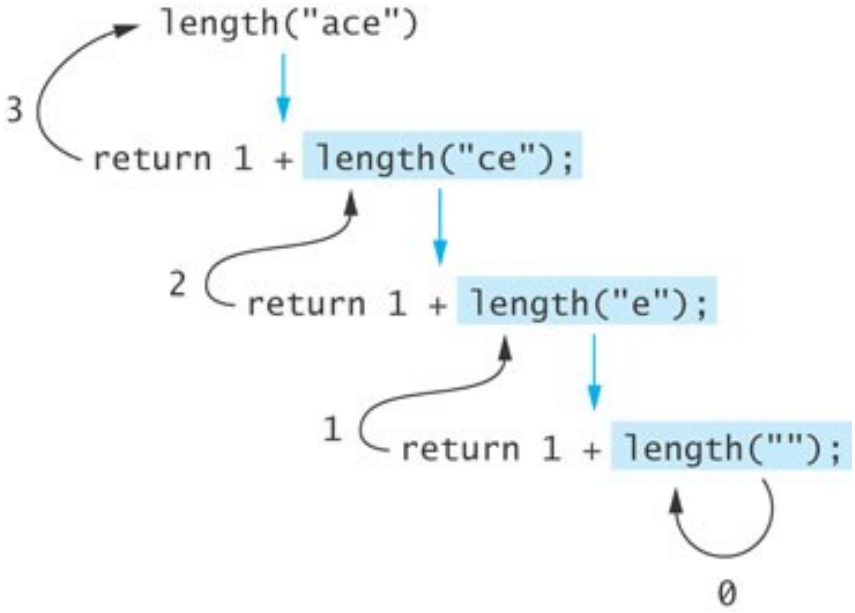
Ukážka – rekurzívne volanie funkcie

- Rekurzívny algoritmus na určenie dĺžky reťazca
  - Ak je reťazec prázdny, výsledok je 0, inak výsledok je (1 + dĺžka reťazca bez prvého znaku)

  - Zdrojový kód:

```
int length(char *s)
{
  if (!s || *s == 0)
    return 0;
  return 1 + length(s+1);
}
```

Krokovanie **`length`**(“ace”):



Vstup (input) – Výstup (output)

- Odkiaľ prídu do programu informácie, čo vlastne chceme programom spracovať?
- Programy chceme robiť všeobecne použiteľné, tak aby vedeli riešiť problémy určitého typu.
- Tzv. vstup môže pochádzať: od používateľa, zo súboru, zo senzorov, zo siete …
- V čase vytvárania programu nevieme aký bude vstup, vieme len aký môže byť (tzv. formát alebo špecifikácia)
- Tzv. výstup (output) sprostredkuje výsledky programu ďalej – na obrazovku, na tlačiareň, do súboru, na sieť (niekomu sa zobrazí web stránka), ...

Načítavanie vstupu

- scanf
- getchar
- fgets

- Čo tieto funckie robia a ako ich môžeme použiť zistíme z dokumentácie ku knižnici stdio.h
- http://www.cplusplus.com/reference/cstdio/scanf/
- http://www.cplusplus.com/reference/cstdio/getchar/
- http://www.cplusplus.com/reference/cstdio/fgets/

Výpis výstupu

- printf
- putchar

- Čo tieto funckie robia a ako ich môžeme použiť zistíme z dokumentácie ku knižnici stdio.h
- http://www.cplusplus.com/reference/cstdio/printf/
- http://www.cplusplus.com/reference/cstdio/putchar/

Píšeme programy...

Úloha: Objem kvádra

- Napíš program v jazyku C, ktorý načíta zo vstupu tri celé čísla do 1000, a vypíše na výstup objem kvádra s rozmermi, ktoré boli na vstupe.

```c
#include <stdio.h>

int main(void)
{
    int x,y,z;
    scanf("%d %d %d", &x, &y, &z);
    printf("%d\n", x*y*z);
    return 0;
}
```

Úloha: Súčet N čísel

- Napíš funkciu **sucet**, ktorá vypočíta súčet čísel v poli: vstupné argumenty pole N čísel (int).
- Postup: výsledok si musím niekde pamätať, označme to ako premennú **vysledok**, potom prejdem všetky čísla a pripočítam ich do premennej **vysledok**
- **Program:**

```
int sucet(int* pole, int n)
{
    int i, vysledok = 0;
    for (i = 0; i < n; i++)
        vysledok += pole[i];
    return vysledok;
}
```

Úloha: Vymeň hodnotu premenných int

- **Napíš (jednorázový) príkaz „niekde“ v programe:**

```
int tmp, u = 10, v = 20;
tmp = u; u = v; v = tmp;
```

- **Napíš (znovupoužiteľnú funkciu) na výmenu hodnôt dvoch premenných typu int.**

```
void swap(int *x, int *y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
// pouzitie
int u = 10, v = 20;
swap(&u, &v);
```

Úloha: Minimum z poľa čísel

- Napíš funkciu **min**, ktorá vypočíta index najmenšieho čísla v poli rôznych čísel: vstup pole N čísel (int).
- Najskôr vymyslíme a napíšeme signatúru funkcie:

```
int min(int* pole, int n)
{
    if (n <= 0)
        return -1;
    int i, x = 0;
    for (i = 1; i < n; i++)
        if (pole[x] > pole[i])
            x = i;
    return x;
}
```

Úloha: Súčet N čísel zo vstupu

- Znovupoužitie programu pre súčet N čísel v poli
- Postup: Pridáme načítanie čísel zo vstupu
- Funkcia **nacitaj_pole**, vstupné argumenty adresa pola (s dostatkom miesta) kam načíta čísla
- **Program**:

```
void nacitaj_pole(int *pole, int *n)
{
    int i;
    scanf("%d", n); // nacitam pocet prvkov pola ... N
    for (i = 1; i <= *n; i++) // nacitam N cisel
        scanf("%d", &pole[i-1]); // i-te cislo do pamate pole[i-1]
}
```

Alokovanie a dealokovanie pamäti

Alokovanie v programovacom jazyku C

- Aké poznáme spôsoby vyhradenia pamäte?
  - **Lokálna premenná**
  - **Gobálna premenná**
  - **malloc/free**

- Dôležitý rozdiel:
  - Globálna premenná je po vyhradení v pamäte **inicializovaná na nuly** – všetky byte sú vynulovaná
  - Lokálna premenná **nie je inicializovaná** (programátor musí naplniť počiatočnú hodnotu)
  - Manuálne vyhradená pamäť (malloc) **nie je inicializovaná** (calloc) je inicializovaná

Alokácia v jazyku C – otázky a odpovede

- Keď si chcem v programe vyhradiť miesto pre int, ako to alokujem?
  malloc(4)
- Keď si chcem vyhradiť miesto pre reťazec s desiatimi znakmi, ako to alokujem?
  malloc(11)
- Keď si chcem vyhradiť miesto pre pole desiatich intov, ako to alokujem?
  malloc(10*4)
  malloc(40)

- Na rôznych platformách môžu mať dátové typy inú veľkosť (32 vs. 64 bit) – operátor **sizeof**
  - Ako vyhradiť miesto pre pole desiatich intov?
    malloc(10*sizeof(int))

Alokácia v jazyku C – príklad

- Napíš program, ktorý malloc-om alokuje miesto pre int, naplní ho konkrétnym číslom (napr. 42) a vypíše ho na obrazovku

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *x = (int*) malloc (sizeof(int));
    *x = 42;
    printf("%d\n", *x);
    return 0;
}
```

Alokácia v jazyku C – príklad

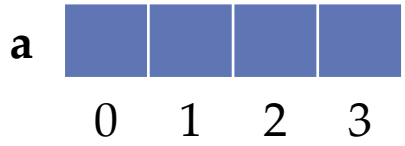
- Napíš program, ktorý malloc-om alokuje miesto pre 10 znakov, načíta do vyhradeného miesta reťazec zo vstupu, a vypíše ho na obrazovku

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char *x = (char*) malloc (11);
    scanf("%s", x);
    printf("%s\n", x);
    return 0;
}
```

Jednorozmerné pole

- Viac rovnakých dátových údajov za sebou
- Rôzne možnosti v programe:
  - Jednorozmerné pole



  - Statické:

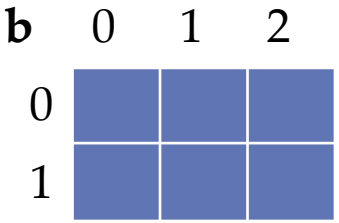
```
int a[4];
```

  - Dynamické:

```
int *a = (int*)malloc(4 * sizeof(int));
```

Viacrozmerne polia

- Dvojrozmerné polia:
  - Obdĺžnikové:



Statické:

```
int b[2][3];
```

Dynamické:

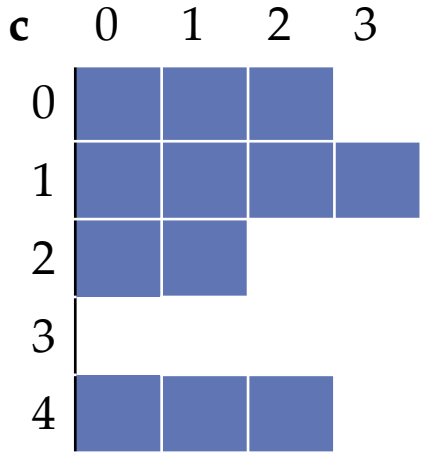
```
int **b = (int**)malloc(2 * sizeof(int*));
b[0] = (int*)malloc(3 * sizeof(int));
b[1] = (int*)malloc(3 * sizeof(int));
```

Viacrozmerne polia

- Dvojrozmerné polia:
  - Zubaté:
    (angl. jagged)

    Statické: nie je možné

    Dynamické:

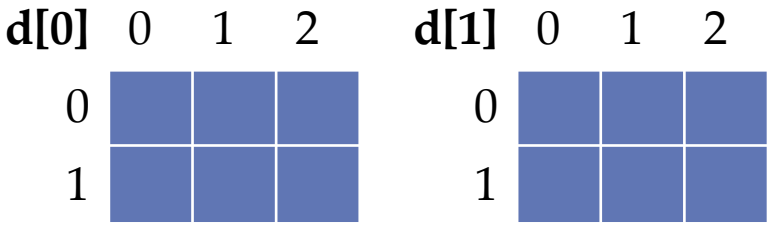



```
int **c = (int**)malloc(5 * sizeof(int*));
c[0] = (int*)malloc(3 * sizeof(int));
c[1] = (int*)malloc(4 * sizeof(int));
c[2] = (int*)malloc(2 * sizeof(int));
c[3] = (int*)malloc(0 * sizeof(int));
c[4] = (int*)malloc(3 * sizeof(int));
```

Viacrozmerne polia

- Trojrozmerné polia:
  - Statické:

```
int d[2][2][3];
```



  - Dynamické:

```
int ***d = (int***)malloc(2 * sizeof(int**));
d[0] = (int**)malloc(2 * sizeof(int *));
d[0][0] = (int*)malloc(3 * sizeof(int));
d[0][1] = (int*)malloc(3 * sizeof(int));
d[1] = (int**)malloc(2 * sizeof(int *));
d[1][0] = (int*)malloc(3 * sizeof(int));
d[1][1] = (int*)malloc(3 * sizeof(int));
```

  - Zubaté – podľa potreby ☺

Kreslíme ASCII

Vzor: Prázdny štvorec

- Nakresli štvorec NxN zo znakov #, vyplnený bodkami.
- N = 5:

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

```
int i, j, n = 5;
for(i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
        if (i == 0 || i == n-1 || j == 0 || j == n-1)
            printf("#");
        else
            printf(".");
    printf("\n");
}
```

- **Tiež dvojitý vnorený cyklus, ale pri výpise znaku rozlišujem či som na okraji alebo vo vnútri**

Vzor: Diagonála

- Nakresli diagonálu zo znakov #, vo štvorci N x N.
- N = 5:

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

```
int i, j, n = 5;
for(i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
        if (i == j)
            printf("#");
        else
            printf(".");
    printf("\n");
}
```

- **Tiež dvojitý vnorený cyklus, ale pri výpise znaku rozlišujem či som na diagonále alebo nie**

Vzor: Štvorec + diagonála

- Nakresli štvorec NxN a diagonálu zo znakov #.
  N = 5:

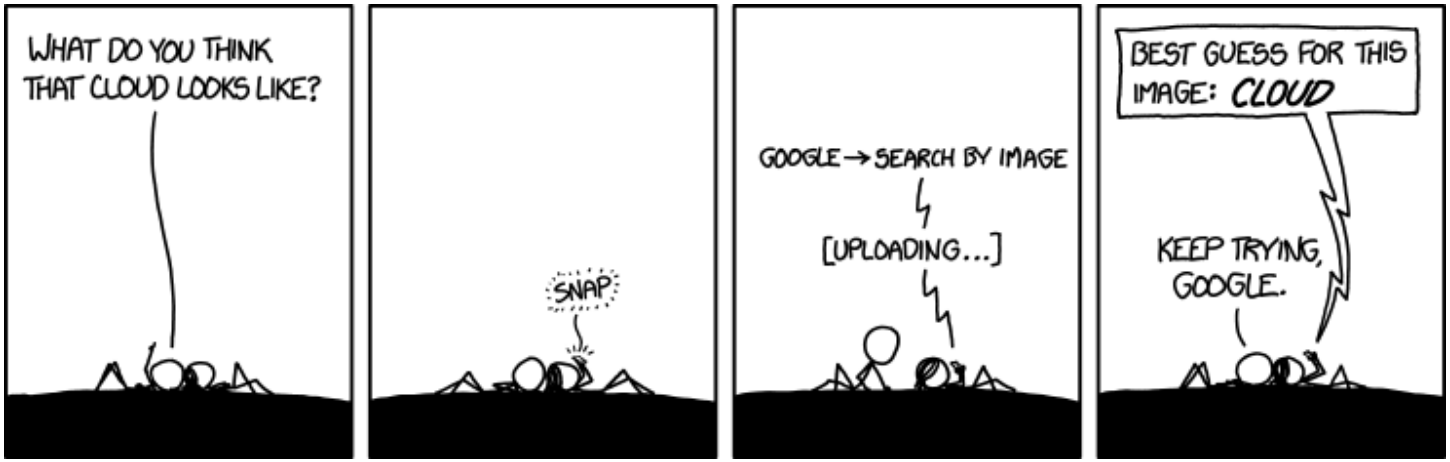
```
#####
##..#
#.#.#
#..##
#####
```

```
int i, j, n = 5;
for(i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
        if (i == 0 || i == n-1 || j == 0 || j == n-1 || i == j)
            printf("#");
        else
            printf(".");
    printf("\n");
}
```

- **Ako by sme skombinovali programy pre nakreslenie štvorca s programom pre diagonálu?**

Najdôležitejšie je pýtať sa otázky...

- https://www.quora.com/What-are-three-things-most-people-don't-know-about-programming
- https://www.quora.com/What-are-some-things-that-programmers-and-computer-scientists-know-but-most-people-dont
- http://programmers.stackexchange.com/questions/185224/what-are-the-programming-concepts-i-should-master-to-have-a-deep-understanding



What are three things most people don't know about programming (1/3) – Aparija Raychaudhury

- **1. Testing and Debugging is a part of programming.**

  Most non-programmers assume they have written correct code. Anyone with any experience of implementing programs work under the assumption that they have written incorrect code.

  Most of my friends have this belief that testing their code is the job of QA engineers and they can just pass their code directly to QA. Well, no, you don't even say you finished the program till you have tested all the boundary conditions, done some profiling, some memory leak testing if your language requires it, tested all failure conditions, etc. THEN the QA Engineers start testing!

What are three things most people don't know about programming (2/3) – Aparija Raychaudhury

- **2. Programming is NOT learning the syntax of a language.**

  It is not even learning a language, a framework, whatever. **It's thinking like computers, knowing how to instruct a 5 year old child to solve differential equations. Oh, did I mention the child has multiple learning difficulties? And, emm, s/he can not speak any language you know!**

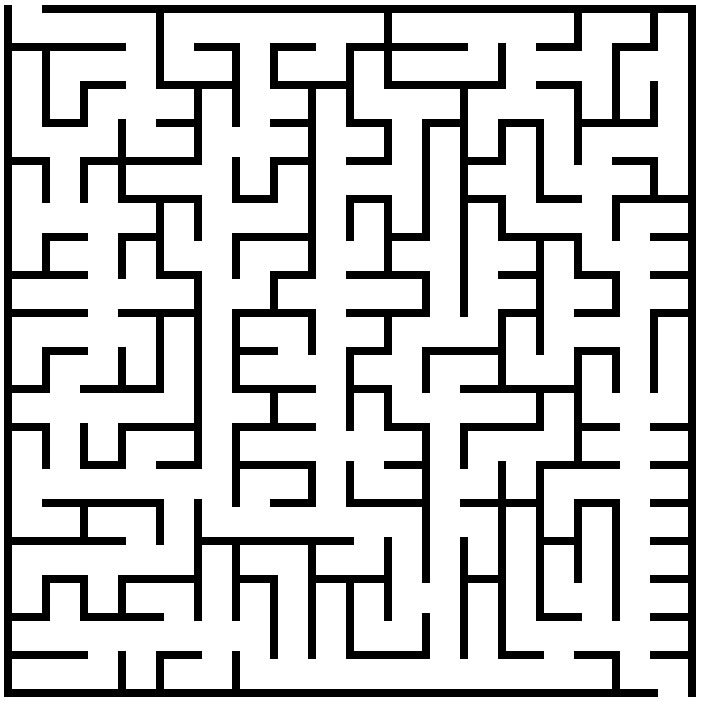
What are three things most people don't know about programming (3/3) – Aparija Raychaudhury

- **3. Knowing a language is not equivalent to knowing it's syntax.**

  Especially engineering students have this faith that they 'know' all the languages taught in the course. Well, no. You know a language when you know it's strengths and weaknesses – what conditions it should be used under; at least one of the different libraries and frameworks that exist; and the syntax. Syntax is not even secondary, it is tertiary.

Bludisko

- Miestnosť – štvorček so stenami naokolo
- Bludisko – 2D pole miestností

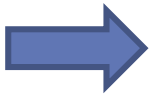


Bludisko: Zadanie projektu

- Dané je bludisko (chodby – znak bodka . a steny – znak mriežka #) **vyznačte prechod z ľavého horného rohu do pravého dolného rohu** znakmi hviezdička *

Bludisko: Ukážka prechodu

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



```
#*################
#*..#.#...#.#...#
#*###.###.#.#.###
#*#.....#..***..#
#*###.#####*#*###
#*****#.#***#*..#
#####*#.#*#.#*###
#.#..*****#.#*#.#
#.#.###.#####*#.#
#...#.#...#..***#
#.###.#.#######*#
#...#.....#....*#
#####.###.#####*#
#...#...#.#.#***#
#.###.#####.#*###
#...........#***#
################*#
```

Bludisko: Ukážka implementácie

- Programujeme spolu …
- Načítanie po riadkoch do obdĺžnikovej matice
- Neznámy počet riadkov (pri každom ďalšom zvyšujeme n++)
- Neznámy počet stĺpcov (m = dĺžka riadku)
- Funkcia hladaj vyznačí cestu z (0,1) do (n-1,m-2)

```
int main(void)
{
    char buf[1000];
    int i, j;

    FILE *f = fopen("bludisko.txt", "rt");
    while(fscanf(f,"%s",buf) > 0)
    {
        m = strlen(buf);
        for (i = 0; i < m; i++)
            map[n][i] = buf[i];
        n++;
    }

    hladaj(0,1, n-1,m-2);

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
            printf("%c", map[i][j]);
        printf("\n");
    }

    return 0;
}
```

Bludisko: Ukážka implementácie (2)

- Opakovane prechádzam celú mapu a ak ešte existuje nepreskúmané políčko susedné od preskúmaného, tak sa tam rozšírim (zaznačím znak X) a zapíšem dĺžku cesty (matica cislo)

```
void hladaj(int sr, int sc, int tr, int tc)
{
    int i,j, k = 47;
    map[sr][sc] = 'X';
    cislo[sr][sc] = 1;
    while (k)
    {
        k = 0;
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            if (map[i][j] == 'X')
            {
                // hore
                if (i-1 >= 0 && map[i-1][j] == '.')
                {
                    cislo[i-1][j] = cislo[i][j]+1;
                  map[i-1][j] = 'X';
                  k++;
                }
                // dole
                if (i+1 < n && map[i+1][j] == '.')
                {
                    cislo[i+1][j] = cislo[i][j]+1;
                  map[i+1][j] = 'X';
                  k++;
                }
```

Bludisko: Ukážka implementácie (3)

- Pokračovanie funkcie hladaj ... (prechod do suseda doľava a doprava).

```
            // dolava
            if (j-1 >= 0 && map[i][j-1] == '.')
            {
                cislo[i][j-1] = cislo[i][j]+1;
              map[i][j-1] = 'X';
              k++;
            }
            // doprava
            if (j+1 < m && map[i][j+1] == '.')
            {
                cislo[i][j+1] = cislo[i][j]+1;
              map[i][j+1] = 'X';
              k++;
            }
        }
}
```

Bludisko: Ukážka implementácie (3)

- Dokončenie funkcie hladaj ... (nakoniec spätne vystopujeme cestu: vyznačím políčka od cieľového využitím matice cislo).

```
i = tr;
j = tc;
while (i != sr || j != sc)
{
    map[i][j] = '*';
    if (i-1 >= 0 && cislo[i-1][j] == cislo[i][j]-1)
        i--;
    else if (i+1 < n && cislo[i+1][j] == cislo[i][j]-1)
        i++;
    else if (j-1 > 0 && cislo[i][j-1] == cislo[i][j]-1)
        j--;
    else if (j+1 < m && cislo[i][j+1] == cislo[i][j]-1)
        j++;

}
map[sr][sc] = '*';
```

Projekt č. 1 – Snehulienka

- Po nedávnej dobrej skúsenosti s jablkami sa Snehulienka rozhodla začať podnikať s ovocím.
- Trpaslíci jej nosia rôzne ovocie, ktoré ona potom predáva.
- Jej úloha v tomto podnikaní je riadiť skladové hospodárstvo...
  - Príjem ovocia
  - Výdaj ovocia

Projekt č. 1 – Snehulienka (2)

- Snehulienka má k dispozícii:
  - Sklad: N políc každá má M pozícii
  - Dve dlhé rolky papiera (A a B), každú tvoria malé útržka, na ktorých si Snehulienka môže niečo poznačiť.
  - Na rolke papiera A si po písmenkách eviduje názvy ovocia
  - Na rolke papiera B si eviduje veľkosť a umiestnenie ovocia v sklade
  - Výbornú vlastnú pamäť, v ktorej si pre každý druh ovocia T v sklade pamätá útržok na ktorom sa na rolke papiera B nachádza záznam o najskôr a najneskôr prijatom ovocí T do skladu

Projekt č. 1 – Snehulienka (3)

- Názvy ovocia si eviduje na dlhej rolke papieru A, na každý útržok si môže poznačiť jedno písmeno
  - Unikátny identifikátor druhu ovocia je index začiatočného písmena na tejto rolke:

0	1	2	3	4	5	6	7	8	9	10	…
k	i	w	i	\0	m	a	n	g	o	\0	…

- Kiwi má identifikátor 0
- Mango má identifikátor 5

Projekt č. 1 – Snehulienka (4)

- Sklad ovocia je N políc pod sebou, každá s M pozíciami

	0	1	2	3	4	5	6	7	8	9	M-1
0	5	5	5	0	0						
1						5	5	5	5		
2								0	0		
N-1											

- Umiestnenie kusov ovocia v sklade si tiež eviduje na dlhej rolke papieru B ... (veľkosť, riadok, stĺpec, ďalší kus)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
2	0	3	8	3	0	0	12	2	2	7	-1	4	1	5	-1	

- Vlastná pamäť, v ktorej si pre každý druh ovocia T v sklade pamätá číslo útržku, na ktorom sa na rolke papiera B nachádza záznam o najskôr a najneskôr prijatom ovocí T do skladu:
  kiwi prvý 0, posledný 8; mango prvý 4, posledný 12

Projekt č. 1 – Snehulienka (4) so šípkami

- Sklad ovocia je N políc pod sebou, každá s M pozíciami

	0	1	2	3	4	5	6	7	8	9	M-1
0	5	5	5	0	0						
1						5	5	5	5		
2								0	0		
N-1											

- Umiestnenie kusov ovocia v sklade si tiež eviduje na dlhej rolke papieru B ... (veľkosť, riadok, stĺpec, ďalší kus)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
2	0	3	8	3	0	0	12	2	2	7	-1	4	1	5	-1	

- Vlastná pamäť, v ktorej si pre každý druh ovocia T v sklade pamätá číslo útržku, na ktorom sa na rolke papiera B nachádza záznam o najskôr a najneskôr prijatom ovocí T do skladu:
  kiwi prvý 0, posledný 8; mango prvý 4, posledný 12

Projekt č. 1 – Snehulienka (5)

- Implementujeme funkcie:

```
 // vrati pocet prijatych
int prijem_ovocia(char *meno, int velkost, int pocet);

// vrati identifikator
int zisti_identifikator(char *meno);

// vypise po pismenkach
void vypis_identifikator(int pos);

// zisti pocet ovocia aspon velkost
int zisti_pocet(char *meno, int velkost_aspon);

// vrati 0 ok, 1 chyba
int vydaj_ovocia(char *meno, int velkost, int pocet);
```

Nabudúce 20.2.

- Algoritmické obchodovanie...