STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Základy procedurálneho programovania 1

Kreslíme ASCII

Pripravte si papier (A4) a písatko!

3. 10. 2016

zimný semester 2016/2017

Programovanie je umenie :)

- **Pripravte si papier (A4) a písatko!**

Ako budeme kresliť?

- Vypisujeme ASCII znaky na obrazovku (printf)
- Budeme kresliť obrázky ohraničené do štvorca
  N riadkov a N stĺpcov
- Základné pravidlo:
  čo už raz vypíšeme, nebudeme prekresľovať
- Na obrazovku vypisujeme **po riadkoch**
- Otázka:
  Môžeme vypísať prvý stĺpec znakov, a až potom druhý stĺpec znakov?

Znaky vypisujeme po riadkoch

- Otázka:
  Môžeme najskôr vypísať druhý riadok znakov, a až potom prvý riadok znakov?
- Nemôžeme.
- Najskôr vypíšeme prvý riadok znakov, potom druhý riadok, potom tretí, a tak ďalej...
- Keď vypisujeme znak, už by mal byť definitívny, nebudeme ho môcť neskôr zmeniť.

Vzor: Plný štvorec

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

```
#####
#####
#####
#####
#####
```

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

- **Dvojitý vnorený cyklus**

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: Trojuholník pod diagonálou

- Nakresli trojuholník pod diagonálou.
- 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");
}
```

- **Aký je rozdiel oproti predchádzajúcemu obrázku/programu?**
  - **Vykresľujeme plochu**

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?**

Všeobecný postup kreslenia

- Vykresľujeme v dvojitom vnorenom cykle
  i pre riadky, j pre stĺpce
- V podmienke kombinujeme jednoduché logické výrazy pre ktoré vypíšeme #, inak vypíšeme .
- Ako vypíšeme čiaru?
  - Logický výraz s rovnosťou (napr. i==n-1 alebo i==j)
- Ako vypíšeme plochu?
  - Logický výraz z nerovnosťou (napr. i >= j)

Vzor: Hviezda

- Nakresli hviezdu NxN zo znakov #.
  N = 7:

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

- **Ako to môžeme rozdeliť na jednoduchšie útvary?**
  - **Kríž (plus) a krížik (x)**

Vzor: Kríž

- Nakresli kríž (plus) NxN zo znakov #.
  N = 5:

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

```
int i, j, n = 5;
for(i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
        if (i == n/2 || j == n/2)
            printf("#");
        else
            printf(".");
    printf("\n");
}
```

- **Dve kolmé čiary v strede útvaru**

Vzor: Krížik

- Nakresli krížik (x) NxN zo znakov #.
  N = 5:

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

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

- **Dve diagonály**

Vzor: Hviezda

- Nakresli hviezdu NxN zo znakov #.
  N = 7:

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

```
int i, j, n = 7;
for(i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
        if (i == n/2 || j == n/2 || i == j || i == n-j-1)
            printf("#");
        else
            printf(".");
    printf("\n");
}
```

- **Spojíme kríž (plus) a krížik (x)**

Vzor: Hviezda vo štvorci

- Nakresli hviezdu vo štvorci NxN zo znakov #.
  N = 7:

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

```
int i, j, n = 7;
for(i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
        if (i == n/2 || j == n/2 || i == j || i == n-j-1 ||
            i == 0 || j == 0 || i == n-1 || j == n-1)
            printf("#");
        else
            printf(".");
    printf("\n");
}
```

- **Spojíme hviezdu a štvorec**

Vzor: Špirála

- Nakresli špirálu NxN.
- N=7:

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

- **Komplikovanejší obrázok – úloha na doma**

STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Základy procedurálneho programovania 1

Základy programovania v jazyku C

zimný semester 2016/2017

3. 10. 2016

Čo sa deje so zdrojovým kódom?

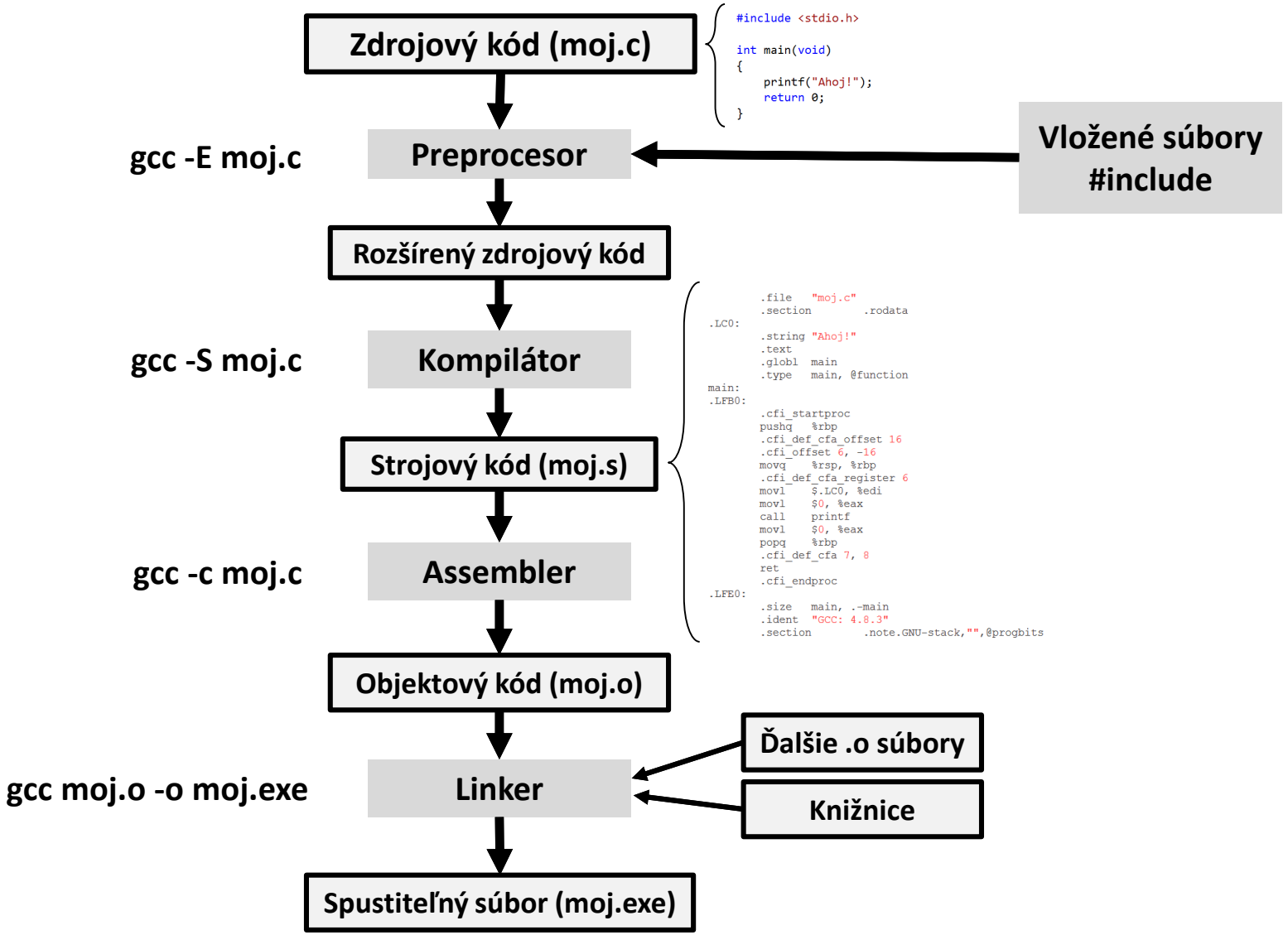
- Prvý program

```c
#include <stdio.h>

int main(void)
{
    printf("Ahoj!");
    return 0;
}
```

- Ako z neho vznikne spustiteľný kód?
  - (.exe súbor vo Windows)
  - (spustiteľný súbor v Unix-e)

Ako vznikne spustiteľný súbor?



Preprocesor

- Spracuje inštrukcie preprocesora (začínajúce #)
  - Vloží obsah súborov **#include**

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

  - Spracuje makrá **#define**
    Nahradí symbolické konštanty:

```
#define MAX 100000

int i = MAX;  // pred spracovanim

int i = 100000; // po spracovani
```

    atď.
- Unix, nástroj **cpp** (gcc -E)

Kompilátor

- Rozšírený zdrojový kód (výstup preprocesora) je skompilovaný do jazyka symbolických inštrukcií (tzv. assembler)
- Čitateľná forma strojového kódu
  - Skrátené názvy (MOV, JMP, MULTP, ...)
  - Pridelenie registrov CPU
  - Prispôsobené pre konkrétnu architektúru (Intel, AMD)
- Unix, nástroj gcc -S

Assembler

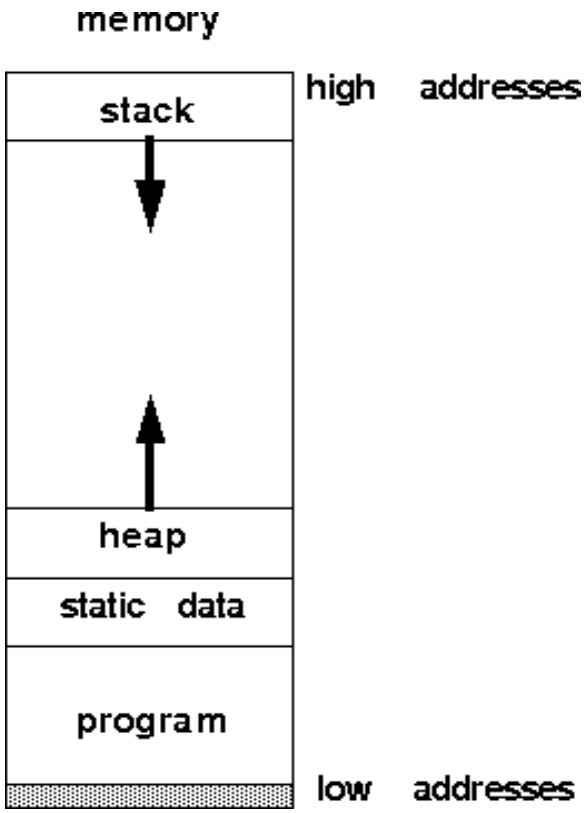
- Program v jazyku symbolických inštrukcií je preložený do objektového kódu (.o / .obj súbor)
  - binárny strojový kód priamo vykonateľný na CPU
  - Každé premenná a inštrukcia dostane pridelené umiestnenie v pamäti – symbolicky alebo cez ofsety
  - Spracuje zoznam všetkých nenájdených odkazov (unresolved references), ktoré nie sú v aktuálnom programe definované, a mali by sa teda nachádzať v iných objektových kódoch
- Unix, nástroj **as** (gcc -c)

Linker

- Spojí objektové kódy do spustiteľného súboru
  - Spojí iné objektové súbory a knižnice do programu
  - Dohľadá vzájomné odkazy (references) funkcií v knižniciach
  - Pridelí absolútne pamäťové umiestnenia všetkým inštrukciám a dátam v programe
- Unix, nástroj **ld**

Loader (spustenie programu)

- Loader (UNIX) vytvorí proces
  - Načítanie súboru
  - Pridelenie adresného priestoru
    program, dáta, zásobník, halda
  - Skok na prvú inštrukciu
    Page fault a prvé načítanie z pamäti
- (Windows)
  Vyhľadať dynamické knižnice
- Zásobník
  - lokálne premenné, volania funkcií
  - Pamäť je pridelená klesajúc
    od najvyššej adresy
- Halda
  - Globálne premenné, alokované premenné



Lexikálne prvky jazyku C

- Tokenizácia: Lexikálny analyzátor číta lexémy a priraďuje im význam
- Token – kategorizovaný blok textu
- Typy tokenov v zdrojovom kóde jazyku C:
  - **Kľúčové slová** (keywords)
  - **Identifikátory** (identifiers)
  - **Konštanty** (constants)
  - **Operátory** (operators)
  - **Separátory** (separators)

Lexikálne prvky: Identifikátory

- Postupnosti znakov používané pre označenie premenných, funkcií, dátových typov, a makier preprocesora
- Identifikátory môžu obsahovať **písmená**, **desiatkové číslice** a **podtrhovník '_'**
  - Prvý znak nemôže byť číslica
  - Odlišujeme veľké a malé písmena:
    meno a MENO sú dva rozličné identifikátory
  - Dobré identifikátory:
    _ahoj, **i2**, **pqr**, **meno**, **meno_stare**, **menoNove**
  - Zlé: **3i, i^3, meno-stare, 2, *, @, milan//, 4poradie, _**
  - Rezervované kľúčové slové (**if**, **int**, **return**, ...) nepoužívať!

Lexikálne prvky: Identifikátory (2)

- Rôzne identifikátory môžu odkazovať na rovnaký kus pamäte, napr. ak by **x** aj **y** odkazovali na rovnaký kus, tak priradenie využitím **x** by „upravilo“ hodnotu v **y**
- **Pri priradení neupravujeme premennú, ale upravujeme dáta v pamäti, ktorá je previazaná s premennou!**
- Teda upraviť dáta môžeme prostredníctvom ľubovoľného identifikátora, a vždy sa upraví to isté pamäťové miesto

Rozsah platnosti (scope)

- Rozsahy platnosti sú viacerých typov
  1. **Blok** – rozsah platnosti v rámci bloku vo funkcií
  2. **Funkcia** – rozsah platnosti v rámci volania funkcie (resp. v rámci bloku tela funkcie)
  3. **Globálny** – rozsah platnosti vo všetkých funkciách

- Čo keď v rozličných rozsahoch platnosti je premenná rovnakého názvu?
  Ak je x v globálnom rozsahu, a aj nejaké nové x deklarované vo funkcií, tak sa vo funkcii vyhradí nová pamäť pre x v rozsahu platnosti funkcie (a do globálneho x nie je možné prostredníctvom mena 'x' zapísať') (podobne ako pri rekurzii)

Lexikálne prvky: kľúčové slová

- Vyhradené identifikátory pre použitie ako súčasť v programovacom jazyku
- Podľa štandardu ANSI C89:
  auto break case char const continue default do double else enum extern float for goto if int long register return short signed sizeof static struct switch typedef union unsigned void volatile while
- **Veľa z nich už poznáme!**
  - **Ostatné spoznáme :)**

Lexikálne prvky: konštanty

- Konštanta vždy má svoj typ
  47 – číslo (int), 4.7 – desatinné číslo (double),
  "Ahoj" – reťazec (char *), 'A' – znak (char)
- **Číselné konštanty** (typy short, int, long, long long):
  - ak začínajú 0x – hexadecimálny zápis (sústava 16)
    napr: 0xABCD (číslice 0, ..., 9, A, B, C, D, E, F)
  - inak, ak začínajú 0 (a nepokračujú x) – oktálny zápis (sústava 8)
    napr: 0123, 047
  - inak, desiatkový zápis – int
  - Suffix u alebo U (pre unsigned), a `l` alebo L (pre long integer)
    napr. 47U je konštanta typu unsigned int
  - **Desatinné čísla** (typy float, double, long double):
    4.7, 4., .7, 0.7, 5e2 (500), 5e-2 (0.05), suffix f:10f (float)

Lexikálne prvky: konštanty

- **Znakové konštanty**:
  - Jeden znak v apostrofoch, napr. 'A' je typu int
  - Viac znakov:
    '\\' backslash \,
    '\?' otáznik,
    '\'' apostrof,
    '\"' uvodzovky,
    '\n' nový riadok,
    '\t' tabulátor,
    a iné
- **Reťazce konštanty**
  - Znaky v uvodzovkách

Lexikálne prvky: operátory

- **Operátor** je token ktorý špecifikuje operáciu (sčítanie, odčítanie, atď.) ktorá sa má vykonať nad operandami
- **Operandy** sú typované objekty ako konštanty, premenné a volania funkcií ktoré vracajú hodnotu
- **Unárne operácie** (jeden operand)
  napr. operátor mínus: **x = -3**
- **Binárne operácie** (dva operandy)
  napr. operátor sčítanie: **y = 2+3**
- **Ternárne operácie** (tri operandy)
  operátor podmienka: **z = priestupny?366:365**

- **Výraz (expression)** je tvorený **aspoň jedným operandom a nula alebo viac operátormi:** 47, 2+2, cosine(3.14159)
  - **Zátvorky spájajú výrazy, vnútorné sú vyhodnotené najskôr**
    2*(3+4) je 14

Lexikálne prvky: separátory

- Separátory oddeľujú tokeny
- Biele znaky (whitespace): medzera, tabulátor, nový riadok) sú separátory ale nie sú to tokeny
- Ostatné separátory sú všetko jednoznakové:

  `( ) [ ] { } ; , . :`
- Biele znaky sa ignorujú všade okrem výskytov kde predstavujú súčasť hodnoty reťazca (napr. „Banska Bystrica“) a sú preto v zdrojovom kóde voliteľné – vhodné na zachovanie prehľadnosti

```c
int main(void)
{
  printf("Ahoj!");
  return 0;
}
```

```c
int main(void){printf("Ahoj!");return 0;}
```

Základné dátové typy

- **Už väčšinu poznáme**
  - `char, signed char, unsigned char`
  - `short, short int, unsigned short int`
  - `int, unsigned int`
  - `long, long int, unsigned long int`
  - `float`
  - `double`
- **Niektoré ďalšie**
  - `long long int, unsigned long long int`
  - `long double`

Smerníky

- Premenná je (len) previazanie identifikátora s pamäťou
- 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**
- Keď poznám adresu premennej, tak viem zmeniť dáta na príslušnej adrese (nepotrebujem nato meno premennej)
- Adresu môžem mat uloženú v premennej – **smerník**

```
int *px = &x;
```

Smerníky (2)

- Smerník – premenná, ktorá obsahuje adresu
- Smerník môže mať dátový typ, napr. smerník na int (adresa pamäte v ktorej je vyhradené miesto pre int)
- Dátový typ „smerník na typ“ píšeme s hviezdičkou typ*, napr. **smerník na int** píšeme **int***
- Príklad:

```
int i = 30;
int* p = &i;
```

... i je meno pre pamäť, kde je int

... p je adresa premennej i

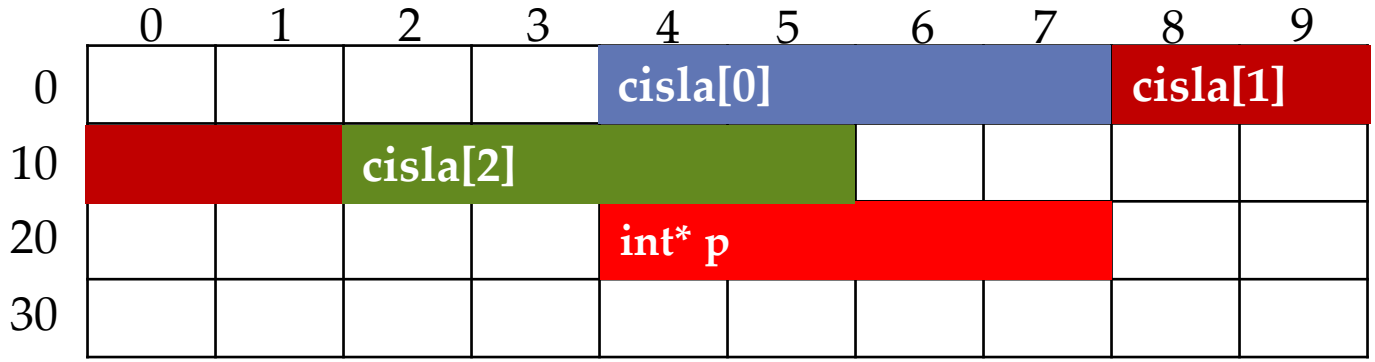
- Pre priradenie využitím smerníka použijeme operátor hviezdička (tzv. dereferencovanie smerníku):
  ***p = 20; je to isté ako i = 20;**

Zložené dátové typy

- Zatiaľ vieme v programe zapísať len jednoduché dátové typy (číslo-int, znak-char, ...)
- Čo ak chceme, aby premenná mohla uschovať viac takýchto dát?
- Viac rovnakých dátových typov za sebov – **pole**
  - množstvo píšeme v hranatých zátvorkách: int cisla[100]
  - k prvkom pristupujeme cez hranaté zátvorky cisla[7], …
  - zvyčajne prvky číslujeme od 0, prvý prvok je cisla[0]

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?

Varianty operátoru priradenia

- Štandardné =

```
int x = 10;
float y = 45.12 + 2.0;
int z = (2 * (3 + funkcia() ));
```

- **Operátor +=** (súčet operandov priradí do ľavého operandu):

```
x += 2; // zvacsi x o 2, rovnako ako x=x+2
```

- Podobne: **-= (odčítanie)**, ***= (násobenie)**, **/= (delenie)**, **%= (zvyšok po delení)**, bitové operácie: **<<=, >>=, &=, ^=, |=**
- Inkrement(post, pre): **x++; ++x;**
- Dekrement (post, pre): **x--; --y;**

Porovnávacie operátory

- Štandardné == vykoná porovnanie rovnosti dvoch operandov
  (hodnoty sa nemenia)
- Operátor != vykoná porovnanie nerovnosti hodnôt dvoch operandov
- Operátory: < (menšie), <= (menšie alebo rovné), > (väčšie), >= (väčšie alebo rovné)

```
if (x == y)
  printf("x je rovne y");
else
  printf("x nie je rovne y");

if (x != y)
  printf("x nie je rovne y");
else
  printf("x je rovne y");
```

Logické operátory

- Logické operátory testujú pravdivostnú hodnotu dvojice operandov
- Operátor && testuje, či majú oba operandy hodnotu pravda
- Operátor || testuje, či má aspoň jeden z operandov hodnotu pravda
- Operátor ! zmení pravdivostnú hodnotu na opačnú

```
if ((x == 47) && (y == 74))
  printf("x je 47 a zaroven y je 74");

if ((x == 47) || (y == 74))
  printf("x je 47 alebo y je 74");

if (!(x == 47))
  printf("x nie je 47");
```

Vedľajší účinok (side effect)

- Reálne viditeľné zmeny behu programu
- Vedľajšie účinky pri vyhodnotení výrazu okrem samotnej výslednej hodnoty výrazu
- Hlavné typy:
  - Upraviť objekt (pamäť)
  - Upraviť súbor
  - Volanie funkcie, ktoré vykoná nejaký z vyššie uvedených vedľajších účinkov
- Sekvenčný bod (sequence point)
  - Bod v programe, ktorý zaručuje, že vedľajšie účinky predchádzajúcich vyhodnotení sú všetky vykonané, a vedľajšie účinky nasledujúcich ešte neboli vykonané
  - Čo vykoná? `i = ++i + 1;`

Príkazy (statements)

- Najmenší prvok zdrojového kódu, ktorý vyjadruje akciu, ktorá sa má vykonať
- Inštrukcia vo vysokoúrovňovom programovacom jazyku, ktorá povie počítaču čo má vykonať
- Blok (zložené príkazy) – nula alebo viacero príkazov zoskupených medzi zloženými zátvorkami { }
- Prázdny príkaz ; (bodkočiarka) – nevykoná nič
- Výraz – pridaním bodkočiarky nakonci vznikne príkaz. 47; 5+5; x && (y > 5), ...
- Riadiace príkazy
  - usmerňovanie toku riadenia programu

Riadiace príkazy (control statements)

- Tie najdôležitejšie už poznáme
  - if
  - for
  - while
  - do-while
  - break
  - continue
  - return

Funkcie

- Funkcia je pomenovaná časť programu, ktorá vykonáva určitú úlohu
- **Argumenty funkcie** sú zoznam pomenovaných dátových typov, ktoré nadobudnú platnosť v rozsahu bežiacej funkcie ako (nové) premenné.
- Napr. výpočet obvodu obdĺžnika:

```
int obvod_obdlznika (int a, int b)
{
  return 2*a + 2*b;
}
```

a,b

return je príkaz, ktorý ukončí funkciu, a ako návratovú hodnotu vráti príslušnú hodnotu.

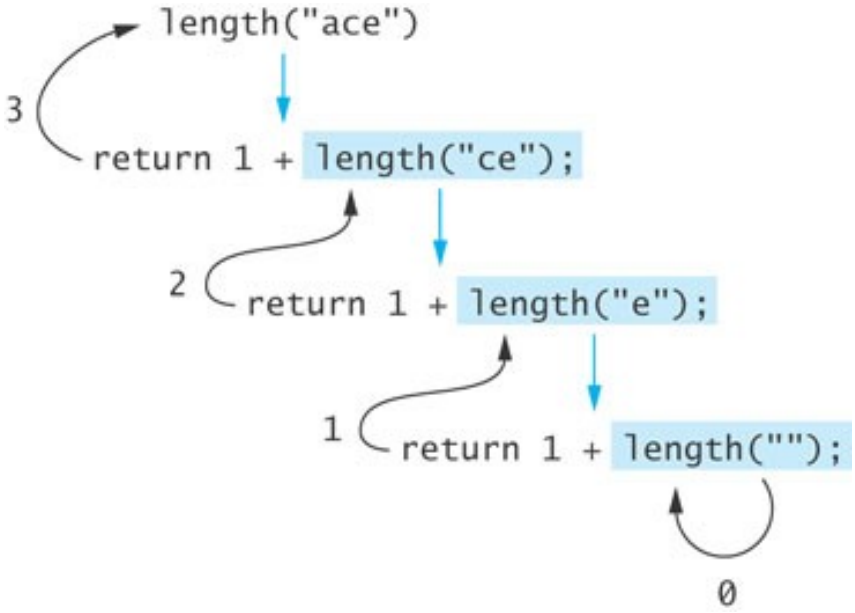
Ukážka – rekurzívne volanie funkcie

- 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

- 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/

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/