

# Procedurálne programovanie

**Ján Zelenka**  
**Ústav Informatiky**  
**Slovenská akadémia vied**



**2023/2024**

# Obsah prednášky

## 1. **Úvod do predmetu**

Podmienky absolvovania

## 2. **Algoritmy a paradigmy programovania**

Procedurálna paradigma

## 3. **Základy programovania v jazyku C**

Kompilovanie programov v jazyku C

Premenné

Terminálový vstup a výstup

Operátory

Aritmetické výrazy

## 4. **Riadiace štruktúry - if**

# Úvod do predmetu

# Ľudia pôsobiaci v predmete

## Prednášky

Ján Zelenka, Ing. Phd. (ÚI SAV) – jan.zelenka@savba.sk

- Bezpečné riadenie procesov (Ing.) na KRIS ZU (2006)
- Automatizácia (PhD.) na KRIS ZU(2009)
- Výskum: automatizácia, optimalizačné metódy inšpirované prírodou  
Európske/národné výskumné projekty a granty, štrukturálne fondy

## Cvičenia

Krammer Peter, Ing. (UISAV)

Považanová, Anna, Ing.

Adamcová Michaela, Bc.

Čopíková Kristína, Bc.

Januška Filip, Bc.

Kuklovský Michal, Bc.

Morháč Dušan, Bc.

Páleník Roman, Bc.

Pirkovský Martin, Bc.

Rudolf Martin, Bc.

Schön Martin, Bc.

Smreček Peter, Bc.

Šalgovič Dominik, Bc.

# Rozvrh

Deň	8.00-8.50	9.00-9.50	10.00-10.50	11.00-11.50	12.00-12.50	13.00-13.50	14.00-14.50	15.00-15.50	16.00-16.50	17.00-17.50
Po				-2.01/a(CPUa) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(1)</sup> <i>M. Rudolf</i>		-2.01/a(CPUa) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(2)</sup> <i>A. Považanová</i>				
				-2.01/b(CPUB) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(3)</sup> <i>A. Považanová</i>		-2.01/b(CPUB) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(4)</sup> <i>R. Páleník</i>				
				-2.01/c(CPUC) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(5)</sup> <i>K. Čopíková</i>		-2.01/c(CPUC) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(6)</sup> <i>M. Pirkovský</i>				
Ut										
St	-2.01/a(CPUa) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(8,11)</sup> <i>A. Považanová</i>	-2.01/a(CPUa) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(7,8)</sup> <i>A. Považanová</i>	-2.01/a(CPUa) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(8,12)</sup> <i>P. Smreček</i>	-2.01/a(CPUa) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(8,9)</sup> <i>M. Kuklovský</i>	-2.01/d(CPUD) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(8,10)</sup> <i>P. Krammer</i>					
	-2.01/b(CPUB) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(8,15)</sup> <i>M. Adamcová</i>	-2.01/b(CPUB) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(8,13)</sup> <i>D. Morháč</i>	-2.01/b(CPUB) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(8,16)</sup> <i>M. Schön</i>	-2.01/b(CPUB) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(8,14)</sup> <i>P. Krammer</i>						
	-2.01/c(CPUC) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(8,18)</sup> <i>F. Januška</i>	-2.01/c(CPUC) (BA-FIIT-FIIT) Procedurálne programovanie <sup>(8,17)</sup> <i>D. Šalgovič</i>								
Št	-1.61 (Aula Magna) (BA-FIIT-FIIT) Procedurálne programovanie <i>P. Čičák</i>									

# Základné informácie o predmete

**Ročník:** 1. ročník 3-ročného bakalárskeho štúdia

**Semester:** zimný 2023/2024

**Odbor:** Informatika

**Trvanie:** 12 týždňov

**Počet hodín týždenne:**

Prednášky: 2

Cvičenia: 2 (povinné)

# Cieľ predmetu

- získať základné znalosti z tvorby algoritmov v rámci procedurálnej paradigmy
- naučiť sa základné konštrukcie jazyka C
- získať zručnosti v tvorbe vybraných algoritmov a programov v jazyku C

# Náplň predmetu

1. základné pojmy jazyka
2. základné údajové štruktúry, smerníky
3. funkcie, iterácia a rekurzia
4. práca so súbormi
5. preprocesor a podmienený preklad
6. polia
7. práca s dynamickým pridelovaním pamäti
8. dynamické štruktúry
9. bitové operácie a polia
10. modulárne programovanie
11. vybrané algoritmy

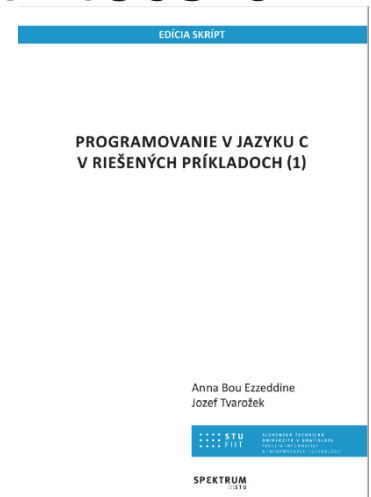


# Kde hľadať informácie?

- Prednášky
- Cvičenia
- Literatúra:
  - Pavel Herout: Učebnice jazyka C, 1. diel, (3./4. vydanie)
  - Programovanie v jazyku C v riešených príkladoch (1), Anna Bou Ezzeddine, Jozef Tvarožek, ISBN 978-80-227-4865-0

iné: odporúčaná literatúra, Internet (?)

<http://www.cplusplus.com/>



# Prednášky

- Prezentácie
- Príklady, programy
- Diskusia so študentami
- Spätná väzba z prednášok a cvičení:  
<https://forms.gle/6q5D2G6UwrtimXEx9>

# Cvičenia

- Riešenie úloh
- Konzultovanie a prezentácia projektu
- **Kontrolovaná aktívna účasť na cvičeniach**
- náhradu ospravedlnenej neúčasti na cvičeniach dohodnúť s cvičiacim

# Cvičenia – Nástroje

Na predmete budeme používať:

- Kompilátor: **gcc** - GNU Compiler Collection (<https://gcc.gnu.org/>)
- Programátorské prostredie: **MS Visual Studio**
  - ako si vytvoriť prvý program [https://www.youtube.com/watch?v=Mb2\\_DQ0BRo](https://www.youtube.com/watch?v=Mb2_DQ0BRo)
- všetko čo odovzdáte, musí byť spustiteľné (odprezentované) na počítačoch v učebniach
- riadte sa pokynmi cvičiacich.

## Alternatívne nástroje:

Kompilovanie

- clang– C language family frontend for LLVM
- pre win\* platformy existujú odvodené prostredia cygwin alebo MinGW

Písanie programov:

- Windows: Dev-C++
- Multiplatformové: Netbeans, Eclipse, Code::Blocks, CLion

# Konzultácie

Predbežne: **individuálne emailom**

- jan.zelenka@savba.sk
- online, alebo na ÚI SAV (miestnosť 202)

## Postup:

- najprv konzultujte so svojim cvičiacim (váš primárny kontakt)
- v prípade problémov s prednášajúcim

**Využite konzultácie už v prípade prvých problémov!**

# Spätná väzba

- anonymný dotazník (prednášky, cvičenia)
- osobne
- anketa predmetu v AIS

Ďakujem Vám za vaše podnety!

## Bodové hodnotenie – Projekty (32%)

### 1. časť projektu (14 bodov)

- zadaná v 5. týždni (17.10.)
- odovzdanie v 7. týždni (6.11. do 23:59)
  - neskoré odovzdanie 8. týždeň (13.11. do 23:59),  
penalizácia => **uznáva sa iba 80% zo získaných bodov**
- za projekt musí získať študent **min. 4 body** (akceptovateľný)

*IDstudenta\_Rok\_projekt\_1.c*

### 2. časť projektu (16 + 2 body)

- zadaná v 9. týždni (14.11.)
- odovzdanie v 11. týždni (4.12. do 23:59)
  - neskoré odovzdanie 12. týždeň (11.12. do 23:59),  
penalizácia => **uznáva sa iba 80% zo získaných bodov**
- za projekt musí získať študent **min. 6 bodov** (akceptovateľný).
- bez bodov za prezentáciu (2 body)

*IDstudenta\_Rok\_projekt\_2.c*

## Bodové hodnotenie – Aktivita (6%)

- študent môže získať celkovo za aktivitu na cvičeniach max. 6 body, min. -2 body
- počas jedného cvičenia môže študent získať 0,2-0,7b alebo stratiť max. 1 bod.
- tiež je možné hodnotenie 0 bodov, alebo +/- 0,5 bodu.
- požadujem aby **každá úloha** bola **v samostatnom súbore s príponou \*.c**, pomenovanie súborov je nasledovné **IDstudenta\_Rok\_C\_U.c**  
(napr. 1254154\_2022\_1\_1.c)



## Bodové hodnotenie – počítačový test (14%)

- test pri počítačoch T1 (cca. 60min) max. 5 bodov
  - za test musí získať študent **min. 1.5 body**
- test pri počítačoch T2 (celé cvičenie) max. 9 bodov
  - za test musí získať študent **min. 3 body**

# Hodnotenie študentov

Projekt:	32 bodov (14 + 18)
Počítačové testy:	14 bodov
Aktivita:	6 body
Záverečný test (skúška):	48 bodov
Spolu:	max. 100 bodov
Záverečné hodnotenie:	min. 56 bodov

## Podmienky absolvovania - Výučba

Získanie zápočtu z cvičení (max. 52 bodov):

- nenulový počet bodov študent môže získať len za časti projektu odovzdané najneskôr v stanovených termínoch požadovaným spôsobom
- aktívna účasť na cvičeniach
  - účasť je povinná na každom cvičení
- vypracovanie **oboch** projektov v akceptovateľnej kvalite, odovzdanie podľa harmonogramu a získanie **minimálneho počtu** bodov
- absolvovanie počítačového testu a získanie min. počtu bodov
- celkovo spolu **minimálne 27 bodov.**

# Podmienky absolvovania - Skúška

Podmienky na vykonanie skúšky:

- získanie min. 27b počas semestra
- **dva termíny**: riadny a opravný.
  - ak si študent chce zlepšiť známku z riadneho termínu, pred opravným termínom sa musí vzdať hodnotenia z riadneho.
- Maximálne bodové hodnotenie za skúšku – **48 bodov**
- Minimálne bodové hodnotenie za skúšku – **18 bodov**

Absolvovanie predmetu:

- podľa stupnice STU (aspoň 56% z celkového hodnotenia)

# Akademická bezúhonnosť

- odpisovanie je vedomé prezentovanie cudzej práce ako svoj vlastný výsledok, teda použitie (častí) práce niekoho iného bez jej citovania je považované za **plagiát**
- autor projektu je preto povinný uviesť v práci všetky zdroje informácií, ktoré použil pri vypracovaní projektu
- nedodržanie akademickej bezúhonnosti rieši disciplinárna komisia a má za následok hodnotenie klasifikačným stupňom **FX**
- **nie je podstatné kto od koho opisoval**

# Algoritmy a paradigmy programovania

# Algoritmus

- predpis, metóda alebo technika, ktorá špecifikuje postup úkonov potrebných na dosiahnutie riešenia nejakej úlohy
  - napr. usporiadanie zoznamu mien podľa abecedy
  - napr. recept na koláč
- v informatike: je **jednoznačná, presná a konečná postupnosť operácií**, ktoré sú aplikovateľné na množinu objektov alebo symbolov (čísiel, šachových figúrok, surovín na koláč)
  - počiatočný stav týchto objektov je vstupom
  - ich koncový stav je výstupom
  - počet operácií, vstupy a výstupy sú konečné (aj keď počítame napr. s iracionálnym číslom  $\pi$ )

# Požadované vlastnosti algoritmov

- **Jednoznačnosť:** každý krok musí byť presne definovaný
- **Rezultatívnosť:** po konečnom počte krokov musí prísť k výsledku
- **Správnosť:** výsledok algoritmu je vždy korektný
- **Efektívnosť:** výpočtový čas a priestor majú byť čo najmenšie
  - často protichodné požiadavky, napr. počítať medzivýsledky opakovane (dlhší čas), alebo si medzivýsledky ukladať (viac priestoru)
- **Determinovanosť:** nasledujúci krok je jednoznačne daný



# Paradigmy programovania

Súhrn spôsobov formulácie problémov, metodologických prostriedkov, štandardných metódik rozpracovania...

Procedurálna paradigma:

- procedúry a riadiace štruktúry, napr. cykly, podmienky
- jazyky: C, PASCAL, COBOL

Objektovo-orientovaná paradigma:

- triedy, objekty, dedenie, polymorfizmus,...
- jazyky: Java, Smalltalk, C++

Funkcionálna paradigma

- funkcie, rekurzia,...
- jazyky: LISP, Haskell, Scheme

Logická paradigma:

- logické predikáty, klauzuly, unifikácia, rezolvenca
- jazyk: PROLOG

# Procedurálne programovanie: podrobnejšie

- **program** je konkrétna reprezentácia algoritmu v nejakom programovacom jazyku pomocou postupnosti príkazov
- príkazy predpisujú vykonanie operácií
  - ak neurčí riadiaca štruktúra inak, vykonajú sa tej postupnosti, v akej sú zapísané
- **operácie**
  - definovaná množina operácií (+-\*/)
  - možnosť vytvoriť ďalšie pomocou procedúr - volanie procedúr s parametrami
- **údaje**
  - uložené v pamäťových miestach (premenná, je meno konkrétneho pamäťového miesta)
  - počas behu programu postupne menia obsah (hodnota premennej)

jazykové konštrukcie pre

- vetvenie (napr. príkaz if, case, switch)
- cyklus (napr. príkaz for, while, do-while).

# Úvod do programovania v jazyku C

```
#include <stdio.h>

int main(void)
{
    printf("Hello world");
    return 0;
}
```

# Jazyk C

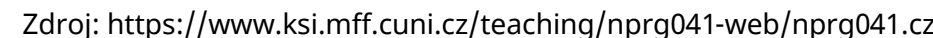
- je univerzálny programovací jazyk nízkej úrovne
  - pracuje len so štandardnými dátovými typmi (znak, celé číslo, reálne číslo...)
- má úsporné vyjadrovanie, je jednoduchý
- je štrukturovaný
- pre mnohé úlohy je efektívnejší a rýchlejší ako iné jazyky
- vďaka štandardu ANSI C sú programy prenositeľné na ľubovoľnú platformu s minimálnymi zmenami zdrojového kódu
- bol navrhnutý a implementovaný pod operačným systémom UNIX

# Obmedzenia jazyka C

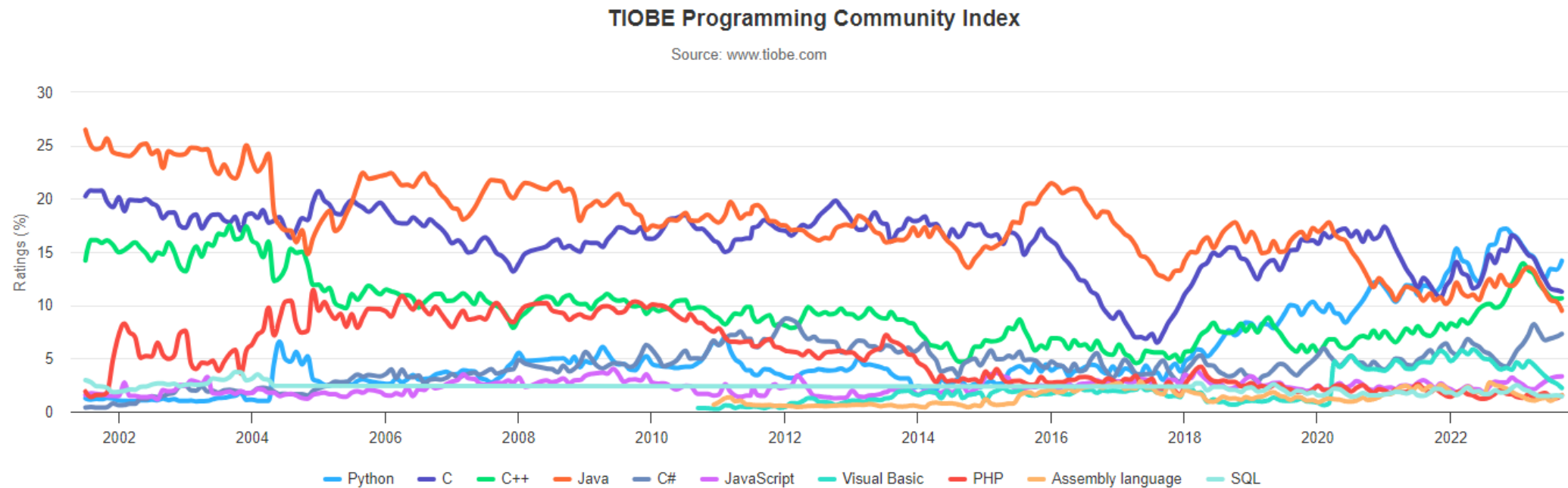
C nepodporuje:

- **Výnimky** – napr. Java
- **Range-checking** (kontrola rozsahu hodnôt -> pole[-567]=5 je v poriadku, nekontroluje pretečenie) – napr. Pascal
- **Garbage collection** (automatickú správu pamäti) – napr. Java
- **Objekty** – napr. C++
- **Polymorfizmus** – napr. Java

## Nové štandardy: C11, C18, C2X



# Použitie jazyka C



# Použitie jazyka C

Vhodný pre:

- Rýchle výpočty (vedecké aplikácie, hry)
- Systémové aplikácie
- Mikrokontrolery (autá, lietadlá)
- Vstavané hardwarové komponenty (telefóny, mikroelektronika)
- Digitálne Signálové Procesory (audio systémy, televízory)

Nevhodný pre:

- Webové aplikácie (PHP, JavaScript)
- Rozsiahle projekty vyžadujúce objektovo-orientovaný prístup (C++, Java)



# Kompilovanie programov v jazyku C

```
#include <stdio.h>

int main(void)
{
    printf("Hello world");

    return 0;
}
```

program.c

```
1001101010110101010101010110
10101010110110101011010101
01010101010101011101001001
01011101101101001011011101
00110101011010101010101101
01010101101101010110101010
10101010101010111010010010
10111011011010010110111010
01101010110101010101011010
```

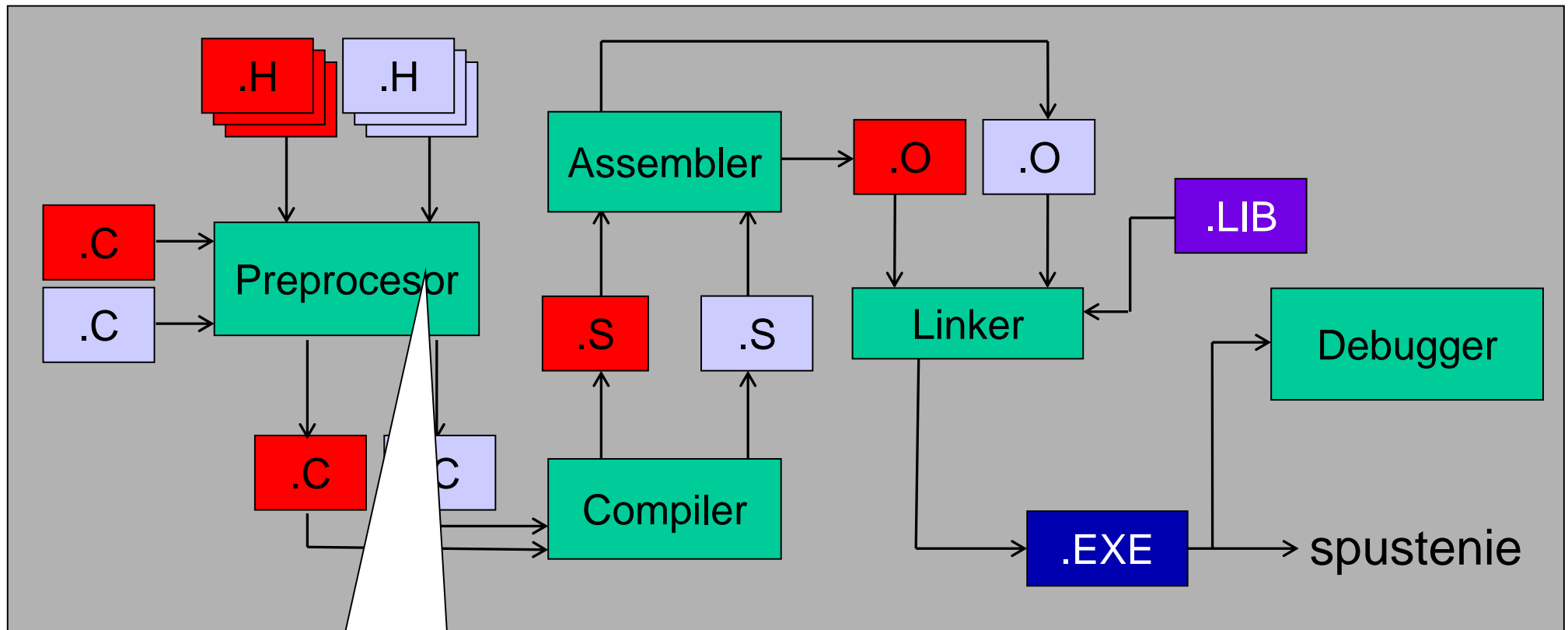
program.out

gcc



stručný manuál nájdete v AISE  
- v zdieľaných dokumentoch  
*Inštalácia GCC.pdf*

# Kompilovanie programov v jazyku C

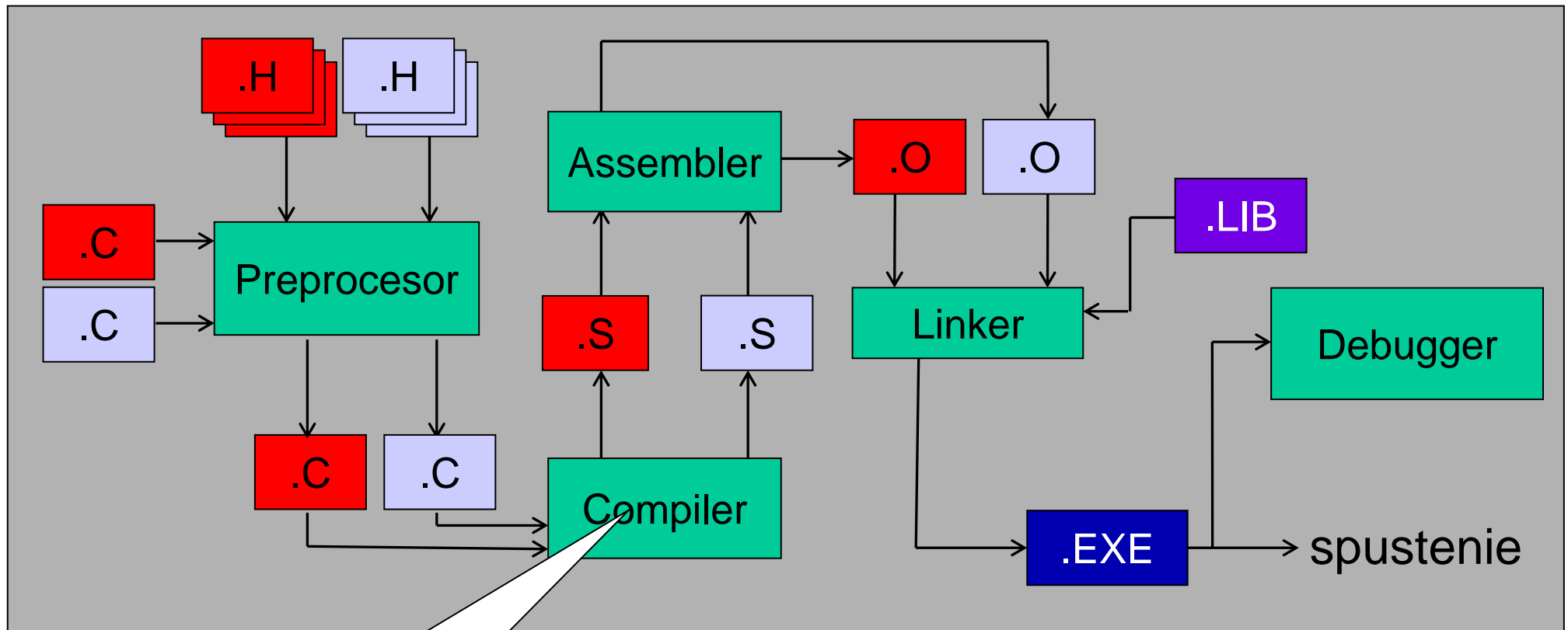


`gcc -E program.c`

## Predspracovanie

- odstránia sa komentáre, všetky odkazované hlavičkové súbory sa vložia do jedného zdrojového súboru
- makra - prispôsobenie prekladu aplikácie kompilačnému prostrediu

# Kompilovanie programov v jazyku C

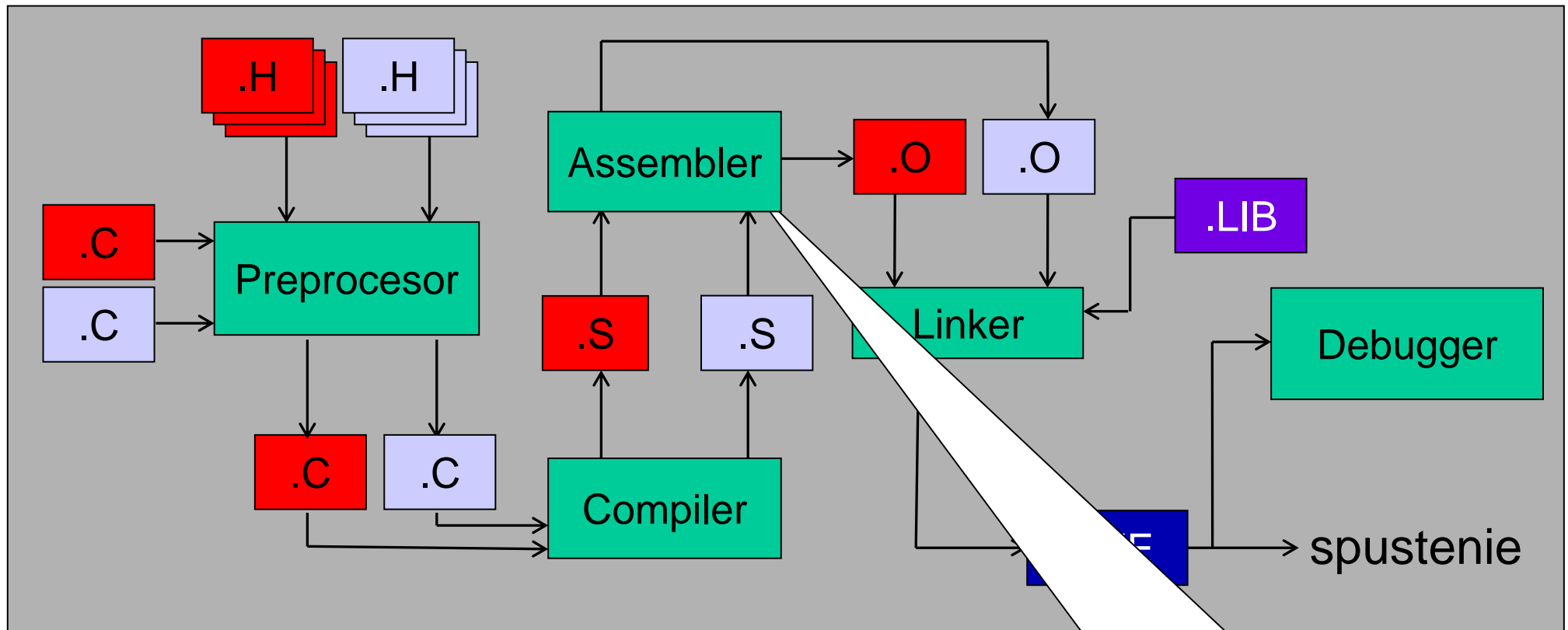


`gcc -S program.c`

## Kompilácia

- prekladá zdrojový kód do assemblerového kódu, kontroluje sa syntax

# Kompilovanie programov v jazyku C

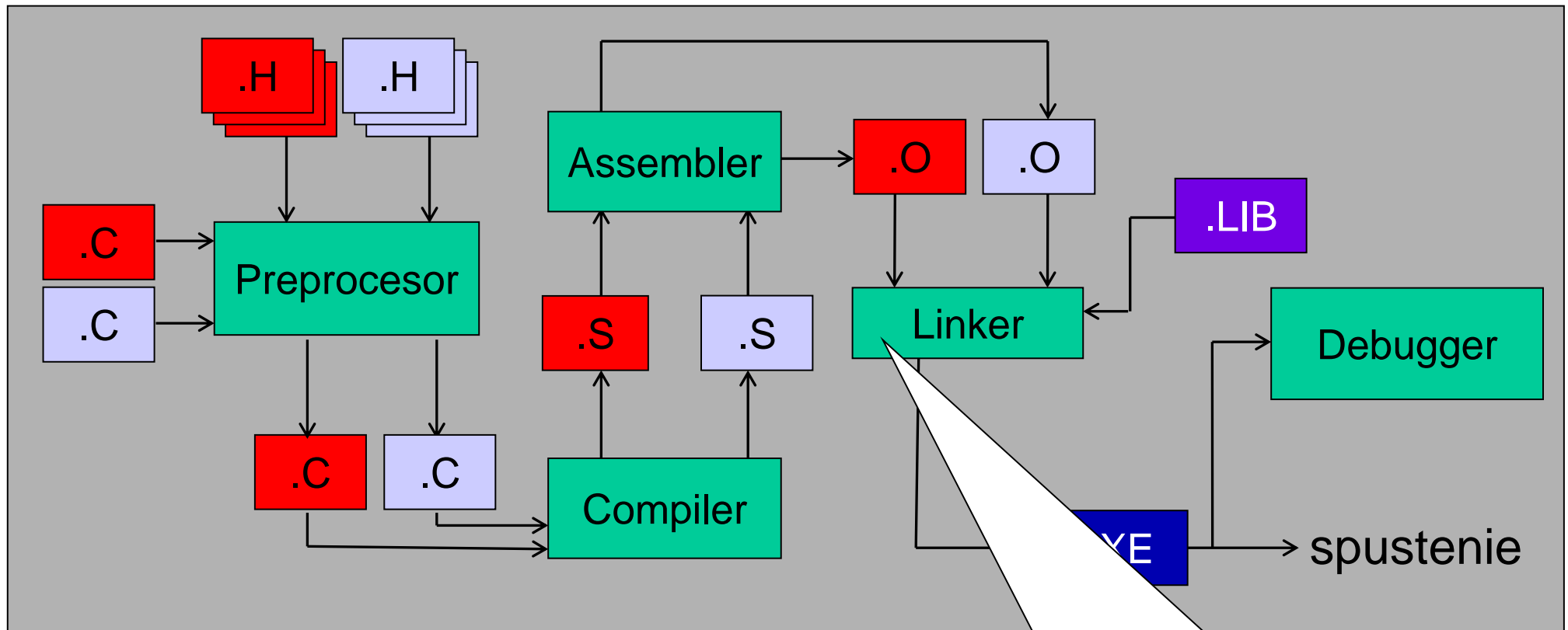


## Zostavovanie

- prekladá assemblerový kód do strojového kódu s relatívnymi adresami premenných, funkcií, atď. t.j. do objektových súborov (10010100100....1000101)
- prezretie obsahu jednotlivých objektových súborov: **objdump -d program.o**

**gcc -c program.c**  
(aj pri Compiler + Assembler)

# Kompilovanie programov v jazyku C



## Linkovanie

- pospája jednotlivé objektové súbory a knižnice (knižnice jazyka C, služby OS...). Relatívne adresy sa nahradia absolútnymi adresami.
- vytvorí sa spustiteľný binárny súbor
  - Zoznam závislostí **ldd program**

## Debugger

- používa sa na ladenie programu

## gcc program.o

(Skompilujem .c na .o súbory nezávisle, v posledom kroku spojím všetky objekty do jedného .exe súboru -> neskôr)

# Zdrojové a hlavičkové súbory

## Zdrojový program .c

- je často potrebné doplniť o vložený súbor (knižnicu)
- jazyk C - **nízkej úrovne a nie** všetko je súčasťou samotného jazyka, ale je definované v knižniciach

## Hlavičkové súbory .h

- zdrojového programu sa \*.h súbory vkladajú, ak program používa funkcie z nejakej knižnice (napr. funkcie na výpis textu na obrazovku)

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

# Komentáre

- slúžia na krátke vysvetlenia častí programu, aby sa v ňom vyznal niekto druhý ale aj vy sami
- sprehládňujú kód

```
/* jednoriadkový komentár */
```

```
/* komentár na  
viacerých riadkoch */
```

- C nedovoľuje vhníezdené komentáre

```
/* komentar v nom /* dalsi komentar */ */
```

# Dátové typy

Dátový typ elementu (premennej, pola...) určuje:

- Hodnoty, ktoré môže daný element nadobúdať
  - napr. **int** môže obsahovať iba celé čísla
- Operácie, sa (ne)dajú vykonávať s daným objektom
  - napr. celé číslo môžeme vynásobiť číslom 5
  - napr. od reťazca nevieme odpočítať číslo 7

C má nasledujúce dátové typy:

- numerické (**int**, **float**, **double**...)
- znakové (**char**)
- smerníkové - *neskôr*
- definované používateľom (**struct**, **union**, **enum**) - *neskôr*



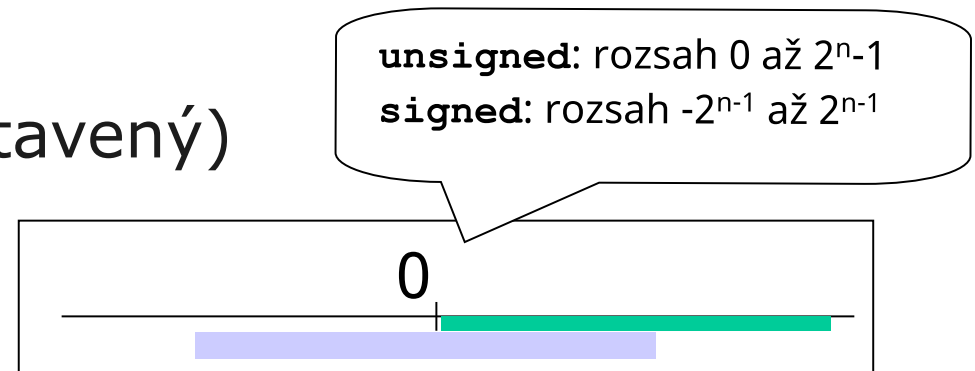
# Primitívne dátové typy

Základné (primitívne) dátové typy sú:

- **char** - znamienkový aj neznamienkový (podľa prekladača)
  - znak dosahuje ASCII hodnoty: 0 - 255
- **int** - znamienkový, celé číslo
- **float/double** – znamienkový, reálne číslo (32 bitov/64 bitov)
- **bool** (logická pravda 1 / nepravda 0, od C99)
  - #include <stdbool.h>
  - v starších verziách sa reprezentuje pomocou typu **int**:
    - FALSE: 0
    - TRUE: nenulová hodnota (najčastejšie 1)

# Modifikátory dátových typov

- `unsigned` – neznamienkový, uvoľnený bit znamienka sa použije na zvýšenie kladného rozsahu hodnôt
- `signed` – znamienkový  
(neuvádza sa, je prednastavený)
- `short` – kratšia verzia typu
  - – `short int (short)`
- `long` – dlhšia verzia typu (ak ju prekladač podporuje)
  - – `long int (long)`
- `long long` – veľmi dlhšia verzia typu (ak ju prekladač podporuje)



# Premenné

- pomenované pamäťové miesto na ukladanie hodnôt
- pred prvým použitím treba špecifikovať dátový typ (definícia premennej)
- každé miesto v pamäti má svoju adresu
- pomocou premennej môžeme čítať alebo zapisovať do danej bunky pamäti
- jazyk C je typový – hodnota premennej musí zodpovedať jej dátovému typu.

# Pomenovanie premenných

Pravidlá pre pomenovanie premenných:

- musí sa začínať znakom anglickej abecedy (a-z, A-Z) alebo \_
  - `_pom` - systémový identifikátor, nepoužívať
- môže obsahovať iba písmená, číslice a podčiarkovník
- rozlišuje veľké a malé písmená (`pom`, `Pom`, `POM`)
- nesmie sa použiť kľúčové slovo jazyka (`int for = 15`)

# Pomenovanie premenných

Odporúčania pre pomenovanie premenných:

- zvolte výstižné meno premennej (`int aaa`)
- meno začínajte malým písmenom
- oddeľujte slová v názve premenných (`pom_x`, `pomX`)

# Definície premenných

- definícia premennej: príkaz, ktorý priradí premennej určitého typu meno a pamäť
- deklarácia premennej: príkaz, ktorý len určuje typ premennej, nepriraduje pamäť -> neskôr

definície:

```
int i;  
char c, ch;  
float f, g;
```

definícia premennej **i** typu **int**

definícia premenných **c, ch** typu  
**char**

definícia premenných **f, g** typu  
**float**

# Inicializácia premenných

- neinicializovaná premenná má **náhodnú hodnotu** (predchádzajúci obsah bunky pamäti)
- premenná sa inicializuje operátorom priradenia

inicializácia premennej **i**

inicializácia v rámci definície

```
i = 6;  
float f = 1.65;  
char c, d, e = 'z', g = 'y';
```

definícia a inicializácia premenných

# Priradenie

Pomocou príkazu `=`, nie ako v Pascale `:=`

## Terminológia:

- výraz: má hodnotu, napr.  $i * 2 + 3$
- priradenie: priradenie hodnoty, napr.  $j = i * 2 + 3$
- príkaz: priradenie ukončené bodkočiarkou,

Niekoľkonásobné priradenie - všetky premenné  $k$ ,  $j$  aj  $i$  budú mať po priradení hodnotu 2

```
k = j = i = 2;
```

napr. 

```
j = i * 2 + 3;
```

príklady príkazov

```
j = 5;  
d = 'z';  
f = f + 3.14 * i;
```



# Globálne a lokálne premenné

globálnu premennú môžu používať v celom programe

```
int i;    /*globalna premenna */
```

```
int main()
```

```
{  
  
  
  
  
  
  
}
```

kučeravé zátvorky - vymedzujú blok

```
    int j; /* lokalna premenna */
```

```
    return 0;
```

lokálnu premennú môže používať len v bloku, v ktorom je premenná definovaná

# Terminálový vstup a výstup

- práca s (bežným) textom
- písmena, číslice, medzery, oddeľovače, zátvorky...
  - Tlačené znaky (ASCII  $\geq 32$ )
- textové dáta sú narezané od binárnych interpretované
  - S bajtom majúcim hodnotu 77 pracujeme ako s písmenom M

# ASCII tabuľka

(American Standard Code for Information Interchange)

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	\$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciichars.com

# Terminálový vstup a výstup

- vstupno/výstupné operácie nie sú časťou jazyka C, obsahuje ich štandardná knižnica
  - dôvodom je, že najviac strojovo závislých akcií je práve vstupno/výstupných - oddeľujú sa nezávislé a strojovo závislé časti jazyka
- popis funkcií na vstup a výstup sa nachádza v hlavičkovom súbore *stdio.h* - pripojíme ho do programu príkazom:

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

# Vstup a výstup znaku

vstup: `int getchar()`

pracujú s premennými  
typu int

výstup: `void putchar(int c)`

Pri volaní `getchar()` píšeme znaky pokým nestlačíme <Enter>. Potom prečíta funkcia prvé písmeno a ostatné ignoruje

## Vstup a výstup iba jedného znaku

# Vstup a výstup znaku: příklad

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

```
int main()  
{
```

```
    int c;
```

```
    c = getchar();
```

```
    putchar(c);
```

```
    putchar('\n');
```

```
    return 0;
```

```
}
```

program přečte znak z  
klávesnice, vytlačí ho a  
odriadkuje

umožní používať funkcie na vstup a  
výstup

načíta znak

vypíše načítaný znak

odriadkuje

# Nový riadok

- `printf ("Hello world! \n Novy riadok");`
  - Nový riadok je vyjadrený špeciálny znakom (escape sekvenciou) `\n`
- Implementácia závisí od OS
  - Unix: `\n` (= posun dolu o jeden riadok)
  - Windows: `\r \n` (= návrat na začiatok riadku, posun dolu o jeden riadok)
- `printf ("Novy riadok \n");`
  - Unix: Novy riadok `\n`
  - Windows: Novy riadok `\r \n`

# Escape sekvencie

Niektoré escape sekvencie majú okrem numerického kódu aj znakový ekvivalent:

\n	0x0A	nový riadok (new line, line feed)
\r	0x0D	návrat na začiatok riadku (carriage return)
\f	0x0C	nová stránka (formfeed)
\t	0x09	tabulátor (tab)
\b	0x08	posun doľava (backspace)
\a	0x07	písknutie (alert)
\\	0x5C	spätné lomítko (backslash)
\'	0x2C	apostrof (single quote)
\0	0x00	nulový znak (null character)



# Formátovaný vstup a výstup

Funkcie, ktoré načítajú celé číslo ako reťazec a prevedú ho do číselnej podoby

Vstup:

```
scanf ("...", ...)
```

Výstup:

```
printf ("...", ...)
```

"..." - formátovací  
reťazec,  
... - premenné

# Použitie scanf()

```
scanf ("%d", &i) ;
```

prečíta celé číslo a  
uloží ho do premennej  
*i*

%d určuje formát  
čítania (dekadické celé  
číslo)

& je nutný - znamená adresu premennej,  
kam sa má uložiť premenná (vynechanie -  
častá chyba)

Ak je načítaných viacero hodnôt, musia byť oddelené bielym znakom

- Medzera, tabulátor...

Pri čítaní sa biele znaky ignorujú!

# Použitie printf()

```
printf("%d", i);
```

na obrazovku vypíše  
hodnotu premennej  
*i*

`%d` určuje formát výpisu  
(dekadické celé číslo)

na rozdiel od `scanf()` sa `&` nepoužíva  
(často sa to mýli)

# Formátovací reťazec

- *scanf()* a *printf()* majú premenný počet argumentov -> formátovací reťazec na určenie počtu a typov premenných
- formátovací reťazec obsahuje:
  - formátovacie špecifikácie - začínajú znakom % a určujú formát vstupu alebo výstupu
  - znakové postupnosti - nezačínajú % a vypíšu sa tak ako sú napísané (používajú sa len v *printf()*)

# Formátovací reťazec

- počet argumentov `scanf()` a `printf()` môže byť väčší ako 2
- počet parametrov musí súhlasiť s formátovacou špecifikáciou
  - počet % = počtu ďalších parametrov
  - ak počty nesúhlasia, kompilátor nehlási chybu, ale program sa nesprávne správa

# Špecifikácie riadiaceho reťazca

za znakom %:

ak načítavame len jeden znak, potom `zn = getchar()` ; je lepšie ako `scanf("%c", &zn)` ;

c	znak
d	desiatkové číslo typu signed int
ld	desiatkové číslo typu signed long
u	desiatkové číslo typu unsigned int
lu	desiatkové číslo typu unsigned long
f	float (pre <code>printf()</code> tiež double)
Lf	long double
lf	double
x	hexadecimálne číslo malými písmenami
X	hexadecimálne číslo veľkými písmenami
o	osmičkové číslo
s	reťazec

pre `printf()` aj  
double

L musí byť veľké

niekedy sa nedá použiť aj pre `printf()`

# Operátory

Operátory definujú ako sa narába s elementami v pamäti

Podľa počtu operandov sa delia na:

- unárne (++ , --)
- binárne (a + b)

Majú rôznu prioritu

- poradie vyhodnocovania
- zmení sa pomocou () (ak si nie ste istý prioritou operátorov, použite zátvorky)

# Aritmetické výrazy

- kombinácia premenných konštánt, operátorov, alebo funkcií
- výsledkom vyhodnotenia výrazu je hodnota s nejakým typom
- aritmetický výraz ukončený bodkočiarkou sa stáva príkazom, napr.

`i = 2`

je výraz s priradením

`i = 2;`

je príkaz

- samotná bodkočiarka je tiež príkaz - nazýva sa prázdny príkaz a využije sa v cykloch



# Unárne operátory

- plus (+)
- mínus (-)
- používanie v bežnom význame

```
...  
x = +5;  
y = -7;  
...
```

# Binárne operátory

- sčítanie (+)
- odčítanie (-)
- násobenie (\*)
- reálne delenie (/)
- celočíselné delenie (/)
- zvyšok po delení celým číslom - modulo (%)

či je delenie celočíselné alebo reálne závisí na type operandov:

int / int      => celočíselné  
int / float    => reálne  
float / int    => reálne  
float / float => reálne

```
int i = 5, j = 13;
```

```
j = j / 4;
```

```
j = i % 3;
```

celočíselné delenie:  $13 / 4 = 3$

modulo: zvyšok po delení  $5 \% 3 = 2$

# Operátory priradenia

okrem jednoduchého priradenia =  
rozšírené prirad'ovacie operátory:

namiesto

```
x = x operator vyraz;
```

kde **x** je l-hodnota, sa použije:

```
x operator= vyraz;
```

```
x += vyraz
```

```
x -= vyraz
```

```
x *= vyraz
```

```
x /= vyraz
```

```
x %= vyraz
```

nedávať medzeru medzi  
operátor a =

a ďalšie, odvodené z iných operátorov

# Relačné operátory

- operátory na porovnanie dvoch operandov (`int`, `double...`)
  - výsledkom je logická hodnota
  - `TRUE`: nenulová hodnota
  - `FALSE`: `0`
- `<`, `>`, `<=`, `>=`, `==`, `!=`
- **`=` VS `==`**
  - `prom = 0` je validný výraz
- pozor na porovnávanie reálnych čísiel
  - obmedzená presnosť
  - nemusia sa rovnať a operátor aj tak vráti `TRUE`

# Logické operátory

- operátory na vyhodnotenie logickej hodnoty výrazu
- logické spojky z výrokovej logiky
  - || - disjunkcia (or, alebo)
  - && - konjunkcia (and, a súčasne);
  - ! -z negácia (not, opak)
- skrátene vyhodnocovanie (Short-circuit evaluation)
  - argumenty sú vyhodnocované zľava a hneď ako je zrejmý konečný výsledok (pravdivostná hodnota výrazu), vyhodnocovanie sa skončí (zapísaný výraz sa teda nemusí vždy vyhodnocovať celý)
  - FALSE && podvýraz, TRUE || podvýraz

```
if ( (1==2) && (hocico ()==1) ) ...
```

Nikdy sa nezavolá!

# Bitové operátory

- $\&$   $\rightarrow$  AND,  $|$   $\rightarrow$  OR,  $\sim$   $\rightarrow$  INVERT,  $\wedge$   $\rightarrow$  XOR
- $\ll$   $\rightarrow$  LSHIFT,  $\gg$   $\rightarrow$  RSHIFT
- Ako logické operátory, na úrovni jednotlivých bitov operandov

0011	0011	0011	
$\&$ 0101	$ $ 0101	$\wedge$ 0101	$\sim$ 0101
= 0001	= 0111	= 0110	= 1010

00000101  
 $\ll 2$   
=00010100

00000101  
 $\gg 2$   
=00000001

Bitový posun je stratový  
ak sa dostane 1 za hranicu dátového typu

# Poradie vyhodnotenia operátorov

1. **++** (inkrement), **--** (dekrement), **!** (logická negácia)
  2. **\***, **/**, **%**
  3. **+**, **-**
  4. **<**, **>**, **<=**, **>=**
  5. **==**, **!=**
  6. **&&** (logická konjunkcia)
  7. **||** (logická disjunkcia)
  8. **=** (priradenie)
- Väčšina operátorov sa vyhodnocuje zľava doprava (sú zľava asociatívne)
- Výnimkou sú: **!**, **++**, **--** a **=**  
(viacnásobné priradenie  $a = b = c$ .)

# Poradie vyhodnotenia operátorov

Aritmetické operátory a operátory porovnania majú väčšiu prioritu ako logické operátory

```
(( c >= 'A' ) && ( c <= 'Z' ) )
```

Zátvorky tam nemusia byť, pretože >= a <= má väčšiu prioritu ako &&

- ak si nie ste istí, či zátvorky dať, radšej ich uveďte
- nezamieňajte && za & a || za | - & a | sú bitové operácie



## Riadiace štruktúry - if

# Vnorené vetvenie

Na jednoduché vetvenie programu môžeme použiť príkaz **if**. Je definovaný v dvoch formách ako:

Neúplný:

```
if (podmienka)
    prikaz
```

ak platí podmienka,  
vykoná sa prikaz

Úplný:

```
if (podmienka)
    prikaz_1;
else {
    prikaz_2;
    prikaz_3;
}
```

ak platí podmienka,  
vykoná sa prikaz\_1,  
inak sa vykoná prikaz\_2  
a prikaz\_3

# Vnorené vetvenie

- ak je v sebe vnorených viac príkazov `if`, tak `else` patrí vždy k najbližšiemu `if`-u
- zjednodušenie cez logické výrazy

```
if (a == 0) {
    if (b == 0) {
        ...
    }
}
```

```
if (a == 0 && b == 0) {
    ...
}
```

# Mnohonásobné vetvenie

```
if (c == 'a')  
    ...  
else if (c == 'b')  
    ...  
else if (c == 'c')  
    ...  
else if (c == 'd')  
    ...  
else  
    ...
```

**Ďakujem vám za pozornosť!**

Spätná väzba: <https://forms.gle/6q5D2G6UwrtimXEx9>