Programování v jazyce C

1 Základy jazyka C

- Historie vzniku a vývoje jazyka C
- Použití, výhody a nevýhody, silné a slabé stránky
- Lexika jazyka ANSI C
- Základy syntaxe, struktura programu v C
- Program typu Hello, world!

Jazyk C a ANSI C
Charakteristika programovacího jazyka

Jazyk C je nízkoúrovňový, víceúčelový programovací jazyk, primárně orientovaný na systémové programování.

- patří do tzv. **algolské skupiny** jazyků (Algol, Pascal, Cobol, PL/1, Ada); **strukturovaný imperativní jazyk**
- procedurální (subrutiny se nazývají **funkce**, ovšem nikoliv ve smyslu funkcionálního programování), veškeré* výkonné příkazy (*statements*) jsou uvnitř funkcí, funkce nemohou být vnořené, návratovou hodnotu lze ignorovat, dovoluje rekurzi
- slabý statický typový systém, lexikální rozsah proměnných
- umožňuje definici uživatelských datových typů
- tradičně spojen s tzv. **preprocesor**em
- významně ovlivnil syntax některých moderních jazyků (Java, JavaScript, C#, PHP, Perl, Rust)

Jazyk C a ANSI C
Historický vývoj jazyka

- vyvinutý v letech 1969 – 1973 v AT&T Bell Labs Brianem Kernighanem a Dennisem Ritchiem, primárně pro potřeby vývoje operačního systému UNIX *{historický exkurs}*
- vychází z jazyka B (a jazyků BCPL a Bon)
- syntax stabilní v roce 1978, Kernighan a Ritchie vydali knihu *The C Programming Language* (de facto standard jazyka C)
- počátek standardizace ANSI roku 1983
- standardizace ukončena ISO roku 1990; jazyk je známý jako **Standard C** nebo **ISO/ANSI C** (vše před 1990 je označováno jako K&R C)
- nejdůležitější definice je **ANSI Standard X3.159-1989** z roku 1989 (většina překladačů „umí” tento dialekt C)
  – této verzi jazyka C se budeme věnovat, protože je nejrozšířenější z normalizovaných verzí (”dobrý základ”)

Jazyk C a ANSI C
Historický vývoj jazyka

- **ISO/IEC 9899:1999** (neformálně **C99**) – předposlední verze standardu jazyka C z roku 1999, která zachytila některá vylepšení, která již v 90. letech 20. stol. běžně implementovala většina překladačů (jako tzv. rozšíření)
  – inline funkce, možnost deklarovat proměnné kdekoliv, nové datové typy (např. `bool` a `complex`), nová komentářová závorka, dynamická pole, atd.
- některé inovace z normy **C99** se však neujaly, překladače je dodnes ignorují (což se snaží vyřešit norma C11)
- **ISO/IEC 9899:2011** (neformálně **C11** či **C1X**) – předposlední vydaný standard jazyka C, není příliš podporován překladači (gcc od verze 4.6 implementuje některé konstrukce)
  – řada kontroverzních prvků jazyka z normy C99 je v C11 označena jako nepovinná/volitelná

Výhody a nevýhody jazyka C

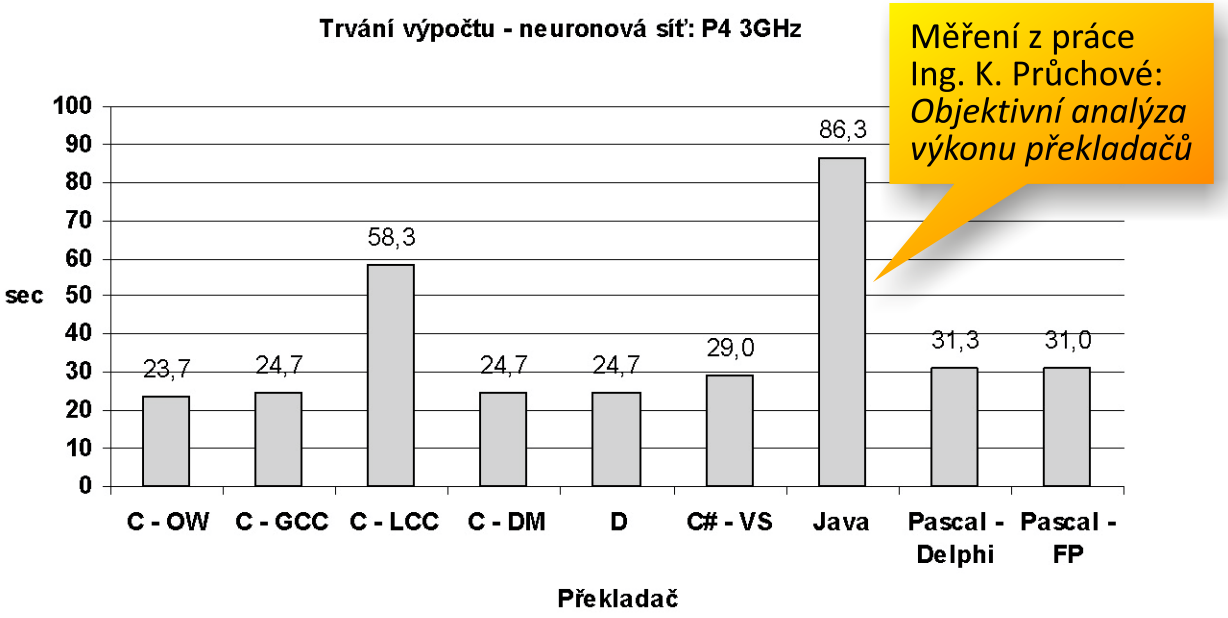
Jazyk C je (nebo může být):

Podporován prakticky na všech výpočetních platformách;
velmi dobře přenositelný mezi platformami (rekompilace);
relativně snadno implementovatelný překladač;
nízkoúrovňový, navržený zejména pro systémové programování (celý UNIX/Linux je v C);
vysoce optimalizovaný binární kód (rychlé vykonávání);
„rychlá" syntaxe, lze rychle psát zdrojový kód;
spolu s Javou absolutně nejrozšířenější prog. jazyk;
standardizovaný (ISO/ANSI).

Velmi špatně čitelný, zkratkovitý, nízkoúrovňový;
nebezpečný, slabá typová kontrola, žádná RT kontrola;
nejednoznačná a nesystematická syntax (`if` – `else`);
nemožnost (jednoduše) testovat bezpečnost kódu.

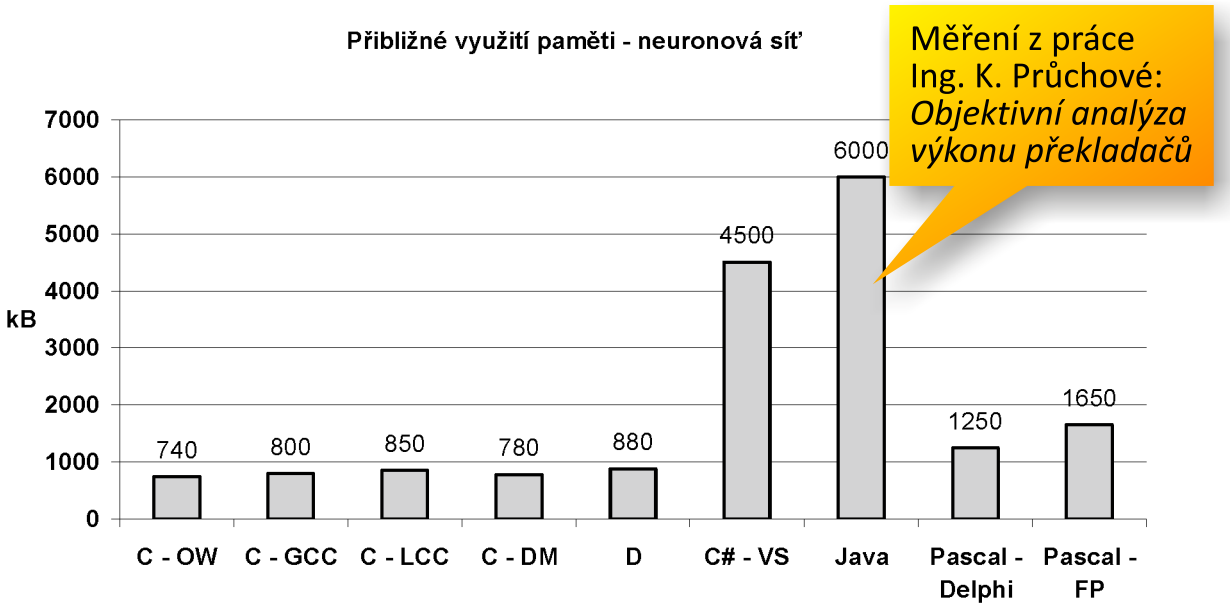
Důvody používání jazyka C
Proč má i dnes smysl se C učit?

C je sice relativně starý jazyk, jehož koncepce je do značné míry překonaná (moderními jazyky, které z jeho syntaxe vzešly, jako např. C++, C# či Java), avšak stále platí, že aplikace naprogramované v C jsou **nepřekonatelně rychlé**:



Důvody používání jazyka C
Proč má i dnes smysl se C učit?

V jazyce C je veškerá paměť v rukou programátora – díky tomu lze ale její využívání značně optimalizovat (také překladače „umí” využívat paměť hospodárněji):



Důvody používání jazyka C
Proč má i dnes smysl se C učit?

Jazyk C stále patří (i přes různé „odborné” prognózy) mezi nejpoužívanější programovací jazyky na světě:

Sep 2019	Sep 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.661%	-0.78%
2	2		C	15.205%	-0.24%
3	3		Python	9.874%	+2.22%
4	4		C++	5.635%	-1.76%
5	6	▲	C#	3.399%	+0.10%
6	5	▼	Visual Basic .NET	3.291%	-2.02%
7	8	▲	JavaScript	2.128%	-0.00%
8	9	▲	SQL	1.944%	-0.12%
9	7	▼	PHP	1.863%	-0.91%
10	10		Objective-C	1.840%	+0.33%
11	34	⇈	Groovy		
12	14	▲	Assembly language		
13	11	▼	Delphi/Object Pascal		
14	16	▲	Go		

TIOBE Programming Community Index September 2019
www.tiobe.com

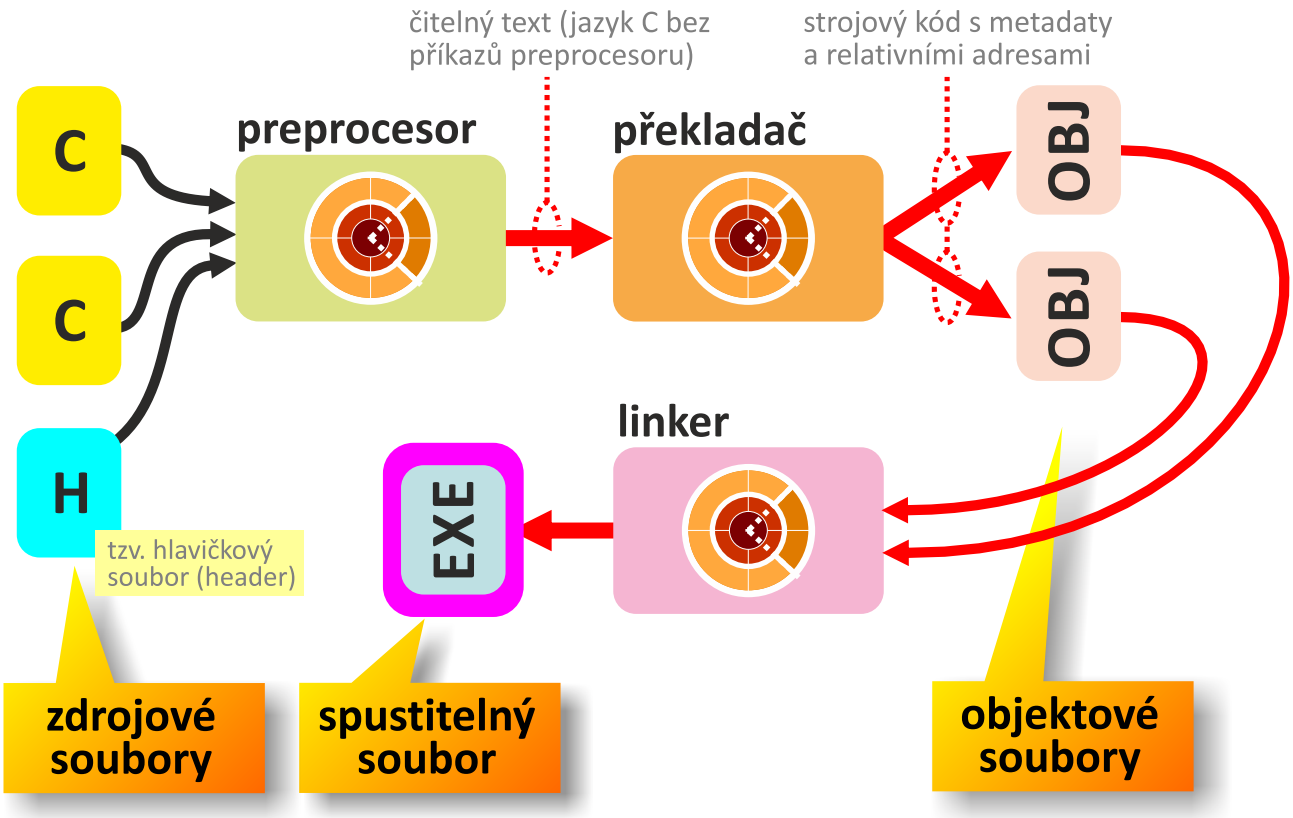
Důvody používání jazyka C
Proč má i dnes smysl se C učit?

Jazyk C (a jeho objektově orientovaný následník C++) **je primárním jazykem pro vývoj operačních systémů**. Tyto OS (resp. jejich jádra) jsou napsány v C/C++:
Microsoft Windows (celá rodina); UNIX → GNU/**Linux**, Android; FreeBSD → MacOS X, iOS; řada dalších (ReactOS, ...)

Řada známých a oblíbených aplikací je napsána v C/C++:

- **proprietární** – Adobe Photoshop & IR, Illustrator, Premiere; Microsoft Office, Visual Studio; Google Earth, Chrome; Mozilla Firefox & Thunderbird; MySQL; Autodesk Maya; Winamp; Oracle JVM; ...
- **open-source** – gcc/MinGW, VLC media player, Inkscape, GIMP, Pidgin, LAME, WireShark, FileZilla, VirtualDub, DC++, Firebird, qTorrent, Code::Blocks, ...

Překlad programu v jazyce C
a sestavení spustitelné aplikace



Doporučené překladače jazyka C
vhodné pro potřeby předmětu KIV/PC

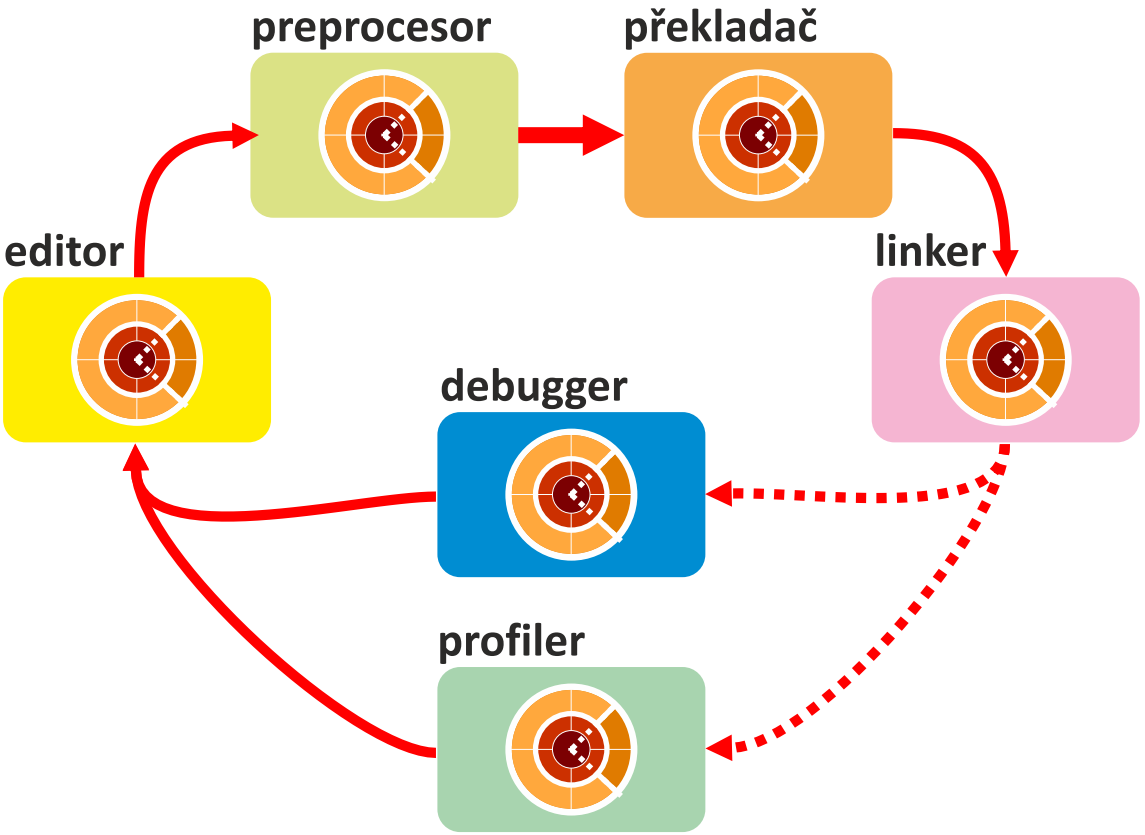
Win32/64 –
Embarcadero Dev-C++ (gcc/MinGW)
Microsoft Visual Studio Community (MS Visual C/C++)
Open Watcom C/C++
Eclipse IDE for C/C++ Developers (CDT) (gcc)
Pelles C
Clang for LLVM

UNIX/Linux –
GNU Compiler Collection (gcc)

- ovládání překladačů viz cvičení, příp. manuál příslušného překladače, webové stránky, atp.
- na http://amos.fav.zcu.cz/pc v sekci Materiály je image DVD s překladači a dalšími nástroji ke stažení

Vývojový cyklus programu v C

Typický postup při vývoji programu



Doporučené debuggery
a další diagnostické nástroje

Debugger –
OllyDbg (jen Win32) – http://www.ollydbg.de/
x64dbg – http://x64dbg.com/
Watcom Debugger, gdb

Analyzátor zdrojového kódu –
splint

Analyzátor binárního kódu –
Dr. Memory – http://www.drmemory.org/
valgrind (jen UNIX/Linux)

Použití valgrindu je naprosto nezbytné – dokáže odhalit tzv. memory leaky (neuvolněnou paměť) a mnoho dalších chyb.

Lexika jazyka ANSI C
Specifika zápisu zdrojového kódu

- zdrojový kód je **prostý textový soubor** (ASCII)
- mezery, tabulátory, CR, LF, atp. překladač ignoruje
- **mezery a tabulátory slouží ke zvýšení čitelnosti**
- nešetřit mezerami, odsazením/tabulátory!
- příkazy se ukončují středníkem *(relikt)*
- **case sensitive – rozlišují se velká a malá písmena!!!**
  (**`id`**, **`Id`**, **`ID`** jsou 3 různé identifikátory)
- všechna klíčová slova a názvy funkcí ze standardní knihovny jsou malými písmeny (tj. žádný camel-case), podtržítko se užívá k oddělení slov ve víceslovných názvech funkcí: **`velmi_dlouhy_nazev_funkce()`**
- v identifikátorech v ANSI C má význam pouze prvních 31 znaků (ostatní překladač ignoruje)
- řetězce v uvozovkách, např.: **`"test"`**, obsah se parsuje

Lexika jazyka ANSI C
Jak funguje lexikální analyzátor C?

- lze použít 52 velkých/malých písmen abecedy
- 10 desítkových číslic
- mezeru, tabulátor, CR/LF (překladač je ignoruje)
- 29 grafických znaků (lze nahradit trojznakem ??x)
- 32 (±) klíčových slov (**malými písmeny**)

- identifikátory mohou být tvořeny písmeny aA – zZ, podtržítkem _ a číslicemi (přičemž číslicí nesmí začínat)
- překladač C zkoumá vnitřek řetězců (později)
- tzv. **žravý** scanner (lexikální analyzátor), tvoří nejdelší možný lexikální atom ze sekvence vstupních znaků

```
x=a+++b;   :   x = a + ++b;   ?   x = a++ + b;
```

Lexika a syntax jazyka C
umožňuje velmi nečitelný zápis

tomtorfs.c

http://www.ioccc.org/

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

int main(int a,char     **A){FILE*B;typedef     unsigned long C;C b
[8]; if(!(a==7&&(B=     fopen(1[A],"rb"))))     return 1;for(7[b]=0
;7[b]<5;7[b]++)b[7[     b]]=strtoul(A[2+7[b     ]],0,16-!7[b]*6);5[
b]=3[b]                 ; while      ((6[b]=    getc(B)
)!=(C)-                 1){if(2      [b])for    (7[b]=0
;7[b]<4                 ;7[b]++      )if(((6    [b]>>7[
b])^(6[                 b]>>(7-7[b])))&1)6[     b] ^=(1
<<7[b])                 ^(1<<(7-7[b]));5[b]     ^= 6[b]
<<(0[b]                 -8);for(7[b]=0;7[b]     <8;7[b]
++)if((                 5[b]>>(0[b]-            1))&1)5
[b]=(5[                 b]<<1)^ 1[b];           else 5[
b]<<=1;                 }5[b]&=((((C)1          <<(0[b]
-1))-1)                 <<1)|1; if(2[b]         )for(7[
b]=0;7[                 b]<(0[b  ]>>1);7        [b] ++)
if(((5[b]>>7[b])^(5     [b]>>(0    [b]-1-7      [b])))&1)5[b]^=((C)
1<<7[b])^((C)1<<(0[     b]-1-7[     b]));5[     b]^=4[b];fclose(B);
printf("%0*lX\n", (     int)(0[     b]+3)>>     2,5[b]); return 0;}
```

- vítěz The International Obfuscated C Code Contest 1998
- ukázka mimořádně nečitelného programu – **semestrální práce takhle nesmí vypadat!**

Lexika a syntax jazyka C
umožňuje velmi nečitelný zápis

dlowe.c

http://www.ioccc.org/

```
#include <stdio.h>
#define PO(o,t)\
(((o>64)&&(o<91))?(((t>96)&&(t<123))?(t-32):(t)):(((t>64)&&(t<91))?(t+32):(t)))

      void main() {                                          char *poo= "poot",
      *Poo="pootpoot"      ,O[9];int      o,t,T,p;(t=p    =0)||(*O='\0');while
      ((o=       getc(   stdin   ))!=(   EOF))if   ((p==   0)&& (((o>64 )&& (
      o<91       )) ||   ((o>     96 )   &&(o<     123)        ))) (
      t!=8       )&&(O   [t]=     o)&&   (O[++     t] =        '\0')
      ;else {if (t>7)    {for     (T =   0 ; T     <=7;        T++ )
      printf("%c",       PO(*(   O+T),   *(Poo+   T)));       printf
      ("%c",              o);}else if     (t>3){for (T        =0;T<=
      3;T++)                                                 printf
      ("%c",                                                 PO(*(O
      +T),*(                                                 poo+T)

) ) ; printf( "%c" , o ) ; } else  printf ( "%s%c" , O , o )  ; ( t =  0 ) || (
* O =  '\0' ) ; ( o == 60 ) && ( ++p ) ; ( o == 62 ) && (p!=0) && ( --p ) ; } }
```

- čitelnost programu zvyšují mezery, prázdné řádky oddělující jednotlivé funkční bloky, odsazení (ify, smyčky, atp.)
- **nešetřit ve zdrojovém kódu bílými znaky**

Imperativní programování
Paradigma imperativního jazyka

Jazyk C je **strukturovaný imperativní jazyk**, tzn.:

- program tvoří posloupnost příkazů, které mění ***stav*** výpočtu
- příkazy nařizují počítači ***vykonat*** určité kroky (akce)
- program popisuje počítači, ***jak*** provést výpočet pomocí posloupnosti elementárních kroků (akcí)

Imperativní paradigma je přirozené, protože hardwarová implementace většiny počítačů je imperativní — procesor vykonává instrukce strojového kódu (posloupnost elementárních akcí). Objektové (či jiné deklarativní) paradigma není hardwarově realizovatelné...

Vzájemný přechod mezi objektovým a imperativním paradigmatem běžně provádí překladač ⇒ programátor to zvládne také... *(tj. znalost Javy neomlouvá*  *)*

Objektové vs imperativní programování
Přechod od OOP k imperativnímu

Základní myšlenka: Ryze objektový program neexistuje. Takový program by pouze popisoval vlastnosti objektů a vztahy mezi nimi ⇒ nic by to nedělalo. Tj. i v objektově orientovaném programu jsou těla metod **imperativní**.

Není třeba se přechodu od objektového paradigmatu k imperativnímu obávat. Stačí vědět, že:

- objektové paradigma ≈ lenost programátora (překladač za něj dělá to, co by mohl udělat ručně – VMT, atp.)
- hardware „chápe" posloupnost příkazů, nikoliv objekty a vztahy mezi nimi ⇒ lepší výkon imp. prog.
- jsem-li „objektový" programátor a chci programovat imperativně, musím si uložit **tužší disciplínu** (klíč k imp. p.)

Objektové vs imperativní programování
Objektová a ne-objektová implementace

JAVA

```
public class Window {
  public String caption;
  private int width, height;
  …

  public Window(String caption) { ... }
  public void show() { ... }
}
```

v OOP metoda sama „ví", s jakým objektem má pracovat (via VMT)

ANSI C

```
struct Window {
  char *caption;
  int width, height;
  …
};

struct Window *create_window(char *capt) { … }
void show_window(struct Window *w) { … }
```

v IP jí to sdělí programátor předáním odkazu/**ukazatele** na stavovou strukturu

Nejjednodušší program v C
Hlavní funkce programu

datový typ návratové hodnoty funkce

funkce **`main()`** – hlavní funkce programu (té předá OS řízení)

demo01.c

```
int main() {
    return 0;
}
```

složené závorky vyznačují složený příkaz – **blok (compound statement)**

návratová hodnota (tu předá program při skončení OS – je v systémové proměnné ERRORLEVEL, resp. $?)

Nejjednodušší program v C
Výsledná podoba po překladu na Intel x86

```
push ebx
push ecx
push edx
push esi
push edi
push ebp
mov  ebp, esp
sub  esp, 4
mov  dword ptr -4[ebp], 0
mov  eax, dword ptr -4[ebp]
leave
pop  edi
pop  esi
pop  edx
pop  ecx
pop  ebx
ret
```

vstup do podprogramu, uložení registrů a příprava zásobníku

návratová hodnota se předává v registru **EAX** procesoru

výstup z podprogramu, obnovení registrů, return

Nejjednodušší program v C
Hlavní funkce programu

demo02.c

```
int main() {
    return 5;
}
```

návratová hodnota z funkce **`main()`** je předána operačnímu systému – lze testovat...

eltest – *UNIX*

```
#!/bin/bash
./demo02
echo $?
```

eltest.bat – *Windows*

```
@echo off
demo02
echo ERRORLEVEL = %ERRORLEVEL%
```

Program HelloWorld v C
Kostra jednoduchého programu

příkaz **preprocesoru** `#include` zajišťuje vložení **hlavičkového souboru (headeru)** knihovny **stdio**

hlavičkový soubor knihovny se vstup/výstup operacemi

helloworld.c

```
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

funkce **printf** (jejíž prototyp je definován v hlavičkovém souboru **stdio.h**) zajišťuje výstup na **stdout**

Předávání argumentů programu z příkazové řádky

```
int main() {
    ...
}
```

program nezpracovává žádné parametry z příkazové řádky OS

počet parametrů, které OS předává programu z příkazové řádky

pole řetězců s jednotlivými parametry (na příkazové řádce odděleny mezerami)

```
int main(int argc, char *argv[]) {
    ...
}
```

Předávání argumentů programu
z příkazové řádky

demo03.c

```
#include <stdio.h>

int main(int argc, char *argv[]) {

    int i;

    for (i = 0; i < argc; i++) {
        printf("%s\n", argv[i]);
    }

    return 0;
}
```

program opíše parametry z příkazové řádky OS na konzoli

Komentáře ve zdrojovém kódu
v ANSI C jen /* ... */

```
int i = 1;    /* do i přiřaď 1 */
```

```
int /* celočíselné */ i = 1;
```

```
/*
   komentář může být i přes více
   řádek, ale nesmí být vnořený!
*/
```

Většina překladačů akceptuje i tento tvar komentáře v C++, ale ten je v rozporu s normou ANSI C:

```
int i = 1;    // tento není dle ANSI
```

Komentáře ve zdrojovém kódu

– znaky uvozující komentář jsou zároveň platné operátory

```
int a, b;
int *c;

a = b/*c;
```

tento **syntakticky správný** přiřazovací příkaz způsobí zakomentování celého zbytku programu

```
a = b / *c;
```

řešení: psát čitelný „řídký” zdrojový kód, operátory oddělovat mezerami

Kostra programu
s vysvětlením jednotlivých částí kódu

znakem **#** začínají vždy **příkazy preprocesoru**

deklarace funkce `main(...)`

demo03.c

```
#include <stdio.h>                          připojení knihoven
int main(int argc, char *argv[]) {
    int i;                                  deklarace proměnných
    for (i = 0; i < argc; i++) {
        printf("%s\n", argv[i]);
    }
    return 0;                               návratová hodnota programu
}                                           začátek/konec složeného příkazu
```

Překlad uvedeného programu
z příkazové řádky

GNU Compiler Collection

```
>gcc demo03.c -o demo03.exe
```

– u **gcc** je nutné uvést za přepínačem **-o** název výstupního spustitelného souboru, jinak vytvoří **a.exe**

Microsoft Visual C/C++

```
>cl demo03.c
```

Open Watcom C/C++

```
>wcl386 demo03.c
```

Překladač Microsoftu i Open Watcom pojmenuje zkompilovaný spustitelný soubor podle předaného zdrojového souboru.