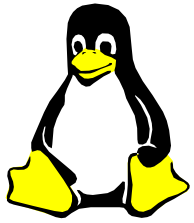
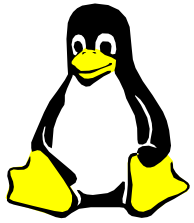


Rychlíkem s tučňákem světem jazyka „C“

Pavel Píša
http://cmp.felk.cvut.cz/~pisa

X35POS 2010
http://dce.felk.cvut.cz/pos

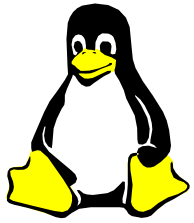
Využité podklady:
Úvod do programovania v jazyku C, Pavel Horovčák, CSc., Igor Podlubný,
http://www.tuke.sk/podlubny/C/Kap2.htm
Jan Kučera, Kurz PB071
http://www.fi.muni.cz/usr/jkucera/pb071/



Historie

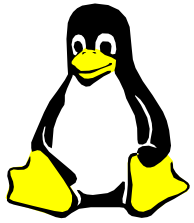
- BCPL (M. Richards, ~1965) -> B (Ken Thompson 1970) -> 1. verze C (Dennis Ritchie). Těsná souvislost s Unixem.
- K-R C: Brian Kernighan, Dennis Ritchie: The C Programming Language (1978)
- ANSI C (**ANSI 89**): Převzata do ISO/IEC.
- ISO/IEC C (**ISO 99**): Aktuálně: ISO/IEC 9899:1999, v r. 2001

  Více o vzniku UNIXu např. v AD4M35OSP
  http://rtime.felk.cvut.cz/osp/prednasky/



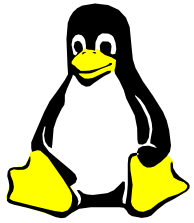
Proč je „C“ stále aktuální

- Svázaný s vývojem konceptu volání Unixového / Posixového i NT jádra (open, read, write, close, ioctl)
- Použitý pro standardní knihovny
- Kompilovaný a vetšinou první portovaný na nové architektury
- Vyšší jazyky (C++, Java, C#, Python) téměř vždy staví na systémových a dalších knihovnách psaných v C. Často použit pro implementaci interpretrů a kompilátorů (Java, Python, Perl, Lisp, atd.)
- Pružná definice základních typů, snadná přenositelnost mezi 8, 16, 32 i 64-bitovými procesory i procesory DSP
- Použit pro implementaci jader a driverů většiny OS, výjimkou např. L4 v C++



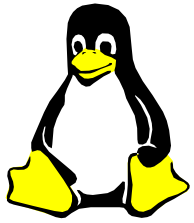
Jazyk „C“ není Java

- Konstrukce pro řízení běhu programu (if, then, else, while, switch,...) jsou v jazyce „C“ v podstatě stejné jako v Javě, ale není zde k dispozici koncept objektů ani automatická správa paměti.
- Hlavičkové soubory (*.h) zvlášť. Knihovny jsou pouze archivy objektových souborů (*.o) bez kontrol typů.
- Systém masek struct/enum/union, které nejsou totéž co nový typ, ten se definuje přes typedef.
- Systém základních typů bez pevně dané bitové délky a reprezentace (char - adresovatelná jednotka, int pro daný procesor nejpřirozenější celočíselný/ordinal typ).
- Pro počty bitů platí, že bitů typu char <= short int <= int <= long int <= long long int. char musí alespoň 8 bitů



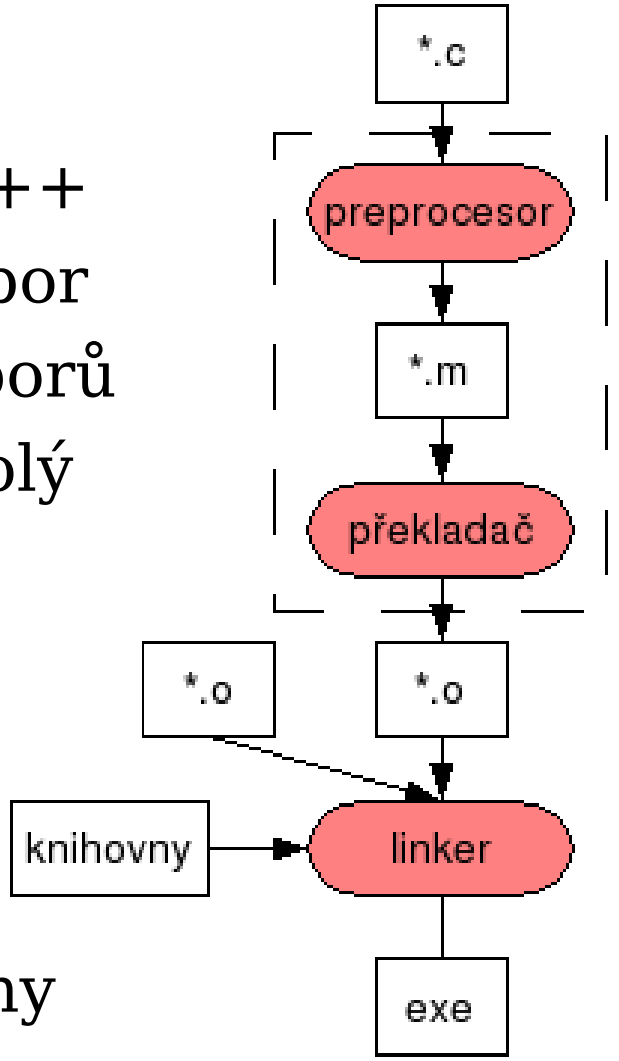
Překladače jazyka C

- Často v kombinaci s překladačem C++.
- DOS/Windows:
  - Borland: Turbo C -> Turbo C++ -> Borland C++ -> C++ Builder
  - Microsoft: Microsoft C -> Microsoft C++ -> Visual C++
- UNIX a multiplatformní překladače
  - GNU: **gcc** (g++ pro jazyk C++)
  - Intel, Metrowerks, IAR, Keil atd

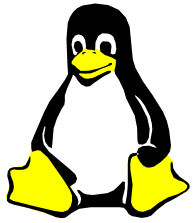


Postup překladu a přípony

- **.c** zdrojový program v C
- **.h** hlavičkové soubory
- **.C .cc .cpp** zdrojový program v C++
- **.o** přeložený objektový/relativní soubor
- **.a** knihovna/archiv objektových souborů
- **.S** assembler s preprocesorem, **.s** holý
- **.so** (Unix), **.dll** dynamické knihovny
  --shared –soname=libx.so
- **.la** Automake/libtool popis knihovny

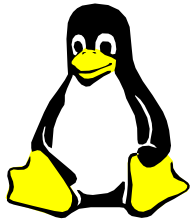



```
gcc -g -c -Wall test.c
gcc -g -c -DASSEMBLY cpuspecific.S
gcc -o test test.o cpuspecific.o -Llibs -lmy
ar  rcs libmy.a funkce_a.o funkce_b.o
```



Volání překladače GCC

- **gcc *přepínače source.c* ...**
  - **-c** pouze překlad na objekt. bez linkování
  - **-g** vytvořit data pro ladicí program (debugger)
  - **-o spustitelný** název výst. soub. (def. **a.out**)
  - **-lknihovna** prohledat knihovnu při linkování
  - **-ansi** , **-std=c89** (ANSI 89) nebo **-std=c99** (ISO 99)
  - **-pedantic** velmi přísná kontrola souhlasu s normou
  - **-Wall** vypisovat při překladu i všechna varování
  - **-O2** optimalizace, **-O3** max. Rychlost, **-Os** min. velikost
  - **-S** překlad na zdrojový kód v assembleru
  - **-E** pouze preprocessor
  - **-v** více informací o překladu
  - **-DMACRO** definice makra



Sestavení pomocí **make**

Makefile

```
#komentáře
MAKRO=definice

cíl: zdroj ...
[TAB]príkaz
[TAB]príkaz
```

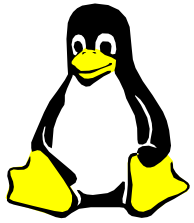
Makra v pravidlech

$@	výstup
$<	první závislost
$^	všechny závislosti
$$	znak „$“

- Pro mnoho úkonů jsou předdefinovaná implicitní pravidla a makra.
  - **CC=cc** překladač jazyka C (lépe **gcc**, pro kříž. překlady **m68k-coff-gcc**, **arm-linux-gcc**, **arm-elf-gcc**)
  - **CFLAGS+= -std=c99 -pedantic -Wall -g**
- Zápis obecných pravidel

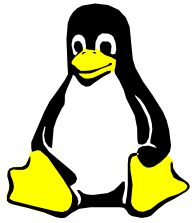
  %.o : %.c

  $(CC) $(CFLAGS) $(CPPFLAGS) $< -o $@



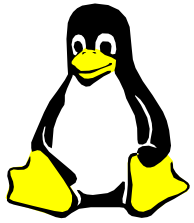
Příklad **Makefile**

```
CC=gcc
CFLAGS=-g -std=c99 -pedantic -Wall

all:prog1

libmy.a: funkce3.o funkce4.o
   ar rcs $@ $^

prog1:libmy.a

prog1: prog1.o funkce1.o funkce2.o
   $(CC) $(CFLAGS) -L. $^ -o $@ -lmy

%.o:%.c
   $(CC) $(CFLAGS) $(CPPFLAGS) -c $<
```



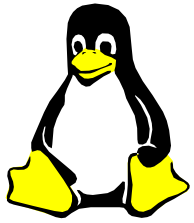
Řešení závislostí a úklid

```
.PHONY : depend dep

dep:
   $(CC) $(CFLAGS) $(CPPFLAGS) -w -E -M *.c \
          > depend

depend:
   @touch depend


clean:
   rm -f *.o *.a prog1
```



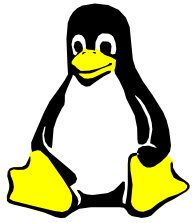
Datové typy v jazyce C

Kategorie	Základní typ	Modifikátory	Příklady konstant
Booleovský	**_Bool** *(jen C99)*		**0** (false) **1** (true)
Celočíselné	**int**	**short/long/** (**long long** *vC99*) **unsigned/signed**	**-5 180L 40000u 3222111000Lu** **037** (=31) **0x1fff** (=8191) (v C99: **-9876543210LL**)
	char	**unsigned/** **signed**	**'A' '\n'** (=10, nový řádek) **'\r'** (=13, zač. řádku) **'\t'** (=9, tab) **'\010'**, **'\b'** (=8, backspace) **'\xff'**
	wchar_t *(jen C99)*	**unsigned/signed**	**L'A' L'\u73c5'** **L'\U2468abcd'**
Reálné	**float**		**1.0f**
	double	**long**	**-1.25 1e-6 -3.5e2950L** (v C99**: 0x1f.8p19**, tj. 31.5*2^{19})
Komplexní *(jen v C99)*	**_Complex/_Imaginary** **float/double/long double** (povinný)		
Prázdný	**void**		



Proměnné a modifikátory

- Jména proměnných obsahují písmena, číslice a „_“.
- Číslice nesmí být na prvním místě
- Prefixy „_ _“ a „_“ jsou konvenčně rezervované pro pomocné funkce kompilátoru a knihoven
- Další modifikátory a kvalifikátory typů a proměnných
  - **const** hodnota nemůže být měněna
    často při předání parametru referencí
  - **volatile** hodnota se mění asynchronně s během kódu
  - **auto** obvyklá lokální proměnná
  - **static** proměnná/fnce. platná pouze v akt. jednotce
  - **extern** pouze deklarace, definice/implementace jinde
  - **register** proměnná v registru, pokud je to možné



Uživatelské typy

Výčtové typy (enum)

```
enum BARVA {CERVENA,MODRA,ZELENA} promenna1;
enum BARVA promenna2;
enum ORIENTACE {SVISLE=1, VODOROVNE=2};
enum ORIENTACE promenna3;
```

Struktury (struct)

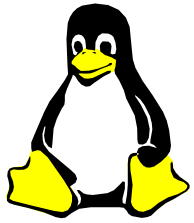
```
struct koule {
  enum BARVA barva;
  int polomer;
};

struct koule koule;
struct koule mic;

struct bod {
  enum BARVA barva;
}
```

Uniony (union)

```
union hvezda {
  struct koule obr;
  struct bod trpaslik;
};
```

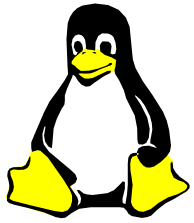
Definice vlastního typu z masky

```
typedef struct koule koule_t;

koule_t zemekoule;
```



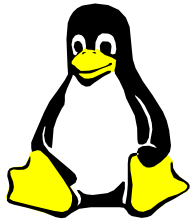
Unární operátory

Symbol	Typ operace	Asociativita
*X	dereference (získání objektu podle adresy)	<-
&X	reference (získání adresy objektu)	<-
-X	aritmetické mínus	<-
!X	logická negace (0 -> 1, nenulové -> 0)	<-
~X	bitový doplněk (po bitech mění 0 na 1 a naopak)	<-
++X	inkrementace hodnoty prom. před vyhodnocením	<-
X++	a po vyhodnocení výrazu ??? X=X++; ++X=X;	->
--X a X--	stejně pro dekrementaci hodnoty proměnné	
([typ])	explicitní převod na typ v závorce (přetypování)	<-
sizeof(X)	získání délky objektu anebo typu (v bytech/adresačních jednotkách)	



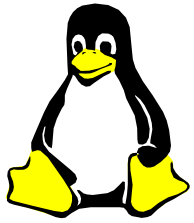
Priorita a binární operátory

Prio.	Symbol	Typ operace	Asociativita
	unární operátory přístup k položce typu „->“ a „.“		
12	* / %	multiplikativní operátory	->
11	+ -	aditivní operátory	->
10	<< >>	operátory posuvu	->
9	< > <= >=	operátory relační	->
8	== !=	rovnost, nerovnost	->
7	&	bitový součin AND	->
6	^	bitový exkluzivní součet XOR	->
5	|	bitový součet OR	->
4	&&	logický součin AND	->
3	||	logický součet OR	->
2	? :	podmíněný operátor	<-
1	= *= /= %= += -= <<= >>= &= |= ^=		<-
	,	čárka (parametry/zapomenutí)	->



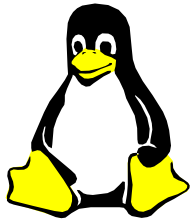
Pole a ukazatele a volání

Symbol	Typ operace	Asociativita
X()	způsobí volání volatelného objektu	->



Jednoduchá funkce

```
int
fibofnc (int n)
{
    if (n <= 0)
        return 0;

    if (n == 1)
        return 1;

    return fibofnc (n - 1) + fibofnc (n - 2);
}
```



Práce s poli a ukazateli

```
int pole[30];
```

deklarace pole prvků

```
int i;
int *p;
```

typ, příslušný ukazatel

```
p=&i;
```

a dereference

```
i=i+1;
*p=*p+1;
i++;
(*p)++;
p[0]++;
```

přístupy

```
p=pole;
p=&pole[0];
```

předání pole

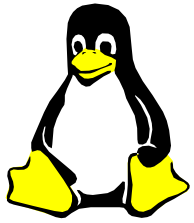
```
for(i=0;i<30;i++)
  pole[i]++;
```

iterace přes prvky

```
p=pole;
for(i=0;i<30;i++){
  (*(p++))++;
}
```

a totéž s posunem ukazatele

```
p++; /* je ekvivalentní */
p=(int*) ((char*)p + sizeof(int));
```



Title

code