



Nízkoúrovňové programování

Proces překladač

Petr Krajča



Katedra informatiky
Univerzita Palackého v Olomouci

- 1 preprocesor vloží hlavičkové soubory, expenduje makra, odstraní komentáře a části kódu nesplňující zadané podmínky
- 2 překladač z kódu v jazyce C vygeneruje kód v assembleru
- 3 kód v assembleru je přeložen do objektových souborů
- 4 objektové soubory jsou sloučeny s použitými knihovnami a je vygenerován spustitelný program

Jednotlivé fáze v překladači gcc

- 1 `gcc -E foo.c //` výstup preprocesoru `cpp` je vypsán na standardní výstup
- 2 `gcc -S foo.c //` je vygenerován soubor `foo.S` obsahující kód v assembleru
- 3 `gcc -c foo.c //` vygeneruje objektový soubor `foo.o`
- 4 `gcc foo.c -o foo //` vygeneruje spustitelný soubor `foo` (bez přepínače `-o` by byl výstupem soubor `a.out`)
- 5 `gcc foo.c -o foo -lm //` přepínač `-l` připojí knihovnu (`-lm` připojí knihovnu obsahující funkce popsané v hlavičkovém souboru `math.h`)



- v C je základní jednotkou pro překlad jeden soubor *.c
- všechny funkce a proměnné, které nejsou označeny jako `static` jsou exportovány (jsou veřejné) a mohou být použity z jiných souborů
- sdílené globální proměnné přes `extern` (problematické)
- označení `static` umožňuje překladači lépe optimalizovat kód
- označení funkcí `inline` indikuje překladači, že nemá použít volání funkce, ale vložit kód na dané místo (užitečné pro malé často používané funkce)



- de facto popis veřejného rozhraní
- definice datových typů
- popis poskytovaných funkcí (ve formě prototypů)
- výjimkou jsou funkce `static inline`
- mohou vzniknout komplexní závislosti
- žádoucí, aby se vyskytoval nanejvýš jednou
- použití preprocesoru

```
#ifndef FOO_H
#define FOO_H
    // kód hlavičkového souboru
#endif
```



- rozdělení překladu na jednotlivé logické části
- vyřešení závislostí
- překlad jen změněných částí
- paralelizace (`make -j X`)
- pomocné funkce (např. odstranění nepotřebných souborů, instalace)

Makefile

- soubor s názvem Makefile
- pravidla pro překlad

```
cil: zdroj-1 zdroj-2 ...  
<TAB>krok prekladu 1  
<TAB>krok prekladu 2
```



prelozi program ze dvou objektovych souboru

```
foo: foo.o bar.o
    gcc -o foo foo.o bar.o
```

```
foo.o: foo.c foo.h
    gcc -c foo.c
```

```
bar.o: bar.c bar.h
    gcc -c bar.c
```

vymaze prelozene soubory

```
clean:
    rm -f foo
    rm -f *.o
```



- bývá nutné (nebo užitečné) překladači nastavit některé parametry
 - -Wall – upozornění na různé problémy v kódu
 - -std – použitý standard jazyka (např. -std=c99)
 - -O – úroveň použitých optimalizací (-O0, ..., -O3)
 - -D – ekvivalent #define

- v Makefile možné nastavit proměnnou, např.

```
CFLAGS = -Wall -O1 -std=c99
```

- následné použití

```
gcc $(CFLAGS) -c foo.c  
$(CC) $(CFLAGS) -c foo.c
```

- předstupěň objektového programování
- modul poskytuje ucelenou funkcionalitu přes jasně definované rozhraní
- hlavičkový soubor obsahuje definice veřejně používaných datových typů, např.

```
struct point {  
    int x, y;  
};
```

- funkce manipulující s těmito strukturami, např.

```
struct point *point_create(int x, int y);  
void point_move(struct point *point, int x, int y);  
void point_print(struct point *point);  
void point_destroy(struct point *point);
```

- prototypy v hlavičkovém souboru
- implementace v samostatném souboru (v případě potřeby může být i více)
- včasná vazba (early-binding), rychlejší kód, nižší flexibilita kódu

- napište modul (datový typ a sadu funkcí) pro manipulaci s maticemi
- vytvořte minimálně funkce, které:
 - vytvoří prázdnou (nulovou matici)
 - přečtou hodnotu na daných souřadnicích
 - změní hodnotu na daných souřadnicích
 - sečtou dvě matice
 - vynásobí dvě matice (volitelná funkce)
 - odmocní matici dvěma (tj. pro každý prvek spočítá jeho druhou odmocninu)
 - uvolní zdroje spojené se zadanou maticí
- pomocí funkce `assert` zajistěte, že jsou provedeny pouze operace s vyhovujícími maticemi a souřadnicemi
- vytvořte program, který bude tento modul používat (vytvoří matice, naplní je daty, provede operace s maticemi a matici vypíše)
- vytvořte `Makefile`
- vyzkoušejte si rychlost programu při zapnutých různých optimalizacích (přepínač `-O`)
- vyzkoušejte si přesunutí funkcí, které čtou/mění jednotlivé hodnoty v matici do hlavičkového souboru (`static inline`)
- měření spotřebovaného času: `time ./foo`