KATEDRA
INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Jazyk C

Práce s preprocesorem, dělení do souborů

Mgr. Markéta Trnečková, Ph.D.

Jazyk C

„Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.“

(Martin Golding)

Zpracování zdrojového kódu

- preprocesor → překladač → linker
- direktivy preprocesoru: #

Příklad

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

Makra

Makra bez parametru (symbolické konstanty)

Příklad

```
o = 2 * 3.14 * r;
```

```
#define JMENO hodnota
```

Příklad (priklad1.c)

```
#define PI 3.14

int main(){
    float r = 3;
    float o = 2 * PI * r;   /* o = 2*3.14*r; */
    return 0;
}
```

```
gcc priklad1.c -E -o priklad1.txt
```

Makra

Makra bez parametru (symbolické konstanty)

Příklad (priklad2.c)

```c
#define JMENO "Marketa"

int main(){
    /* Vypise Moje jmeno je JMENO */
    printf("Moje jmeno je JMENO");
    return 0;
}
```

Jak vypsat konstantu `JMENO`?

Makra

Makra bez parametru (symbolické konstanty)

Předefinování hodnoty

Příklad (priklad3.c)

```
#define JMENO "Marketa"

#undef JMENO

#define JMENO "TRNECKOVA"
```

Dlouhé konstanty

Příklad

```
#define DLOUHA_KONSTANTA 1.23456798\
910111213
```

Makra

Předdefinované konstanty

- `__LINE__` – číslo právě zpracovávaného řádku programu (desítkové číslo),
- `__FILE__` – jméno právě zpracovávaného programu (řetězec),
- `__DATE__` – aktuální datum (řetězec ve tvaru mmm dd yyyy),
- `__TIME__` – čas překladu (řetězec ve tvaru hh:mm:ss),
- `__STDC__` – má hodnotu 1 pokud se jedná o ANSI C .

Makra

Makra s parametry (inline funkce)

```
#define jmeno(arg_1, arg_2, ..., arg_n) telo_makra
```

Příklad (na2.c)

```
#define na2(x) ((x) * (x))
#define na2a(x) x * x

na2(f + g);   /* ((f+g)*(f+g)) */
na2a(f + g); /* f+g*f+g */
```

Makra

Makra s parametry (inline funkce)

Příklad (na2.c)

Co bude výsledkem?

```
#define na2(x) ((x)*(x))

int i = 2;
na2(i++);
```

Makra

Makra s parametry (inline funkce)

Příklad (m.c)

```
#define m(x) ((x) < 0 ? m(-x) : m(x))
```

Příklad (plus.c)

```
#define add(x,y) (x)+(y)
#define plus(x,y) add(x,y)
```

Makra

Operátory # a

- # – nahradí se řetězcem odpovídajícím předanému argumentu
- ## – spojení dvou argumentů v jeden řetězec

Příklad (operatory.c)

```
#define plus(X,Y) printf("%s + %s = %d", #X, #Y, (X) + (Y))
/* plus(3,2);
   printf("%s + %s = %d", "3", "2", (3) + (2));*/

#define var(X) promenna ## X
/* var(10)
   promenna10 */
```

Podmíněný překlad

```
#if podminka
    kod
#endif
```

```
#if podminka1
    cast_1
#elif podminka2
    cast_2

    ...

#else
    cast_n
#endif
```

Podmíněný překlad

Podmíněný překlad řízený konstantním výrazem

```
#if konstantni_vyraz
    cast_1
#else
    cast_2
#endif

/* NEBO */

#if konstantni_vyraz
    cast_1
#elif konstantni_vyraz2
    cast_2
#else
    cast_3
#endif
```

Podmíněný překlad

Podmíněný překlad řízený konstantním výrazem

Příklad (podmineny.c)

```
#define ENG 1

#if ENG
    #define ERROR "error"
#else
    #define ERROR "chyba"
#endif
```

Podmíněný překlad

Podmíněný překlad dle symbolické konstanty

`#ifdef`, `#ifndef`

Příklad

```
#define ENG 1

#ifdef ENG
    #define ERROR "error"
#else
    #define ERROR "chyba"
#endif
```

Podmíněný překlad

Podmíněný překlad dle symbolické konstanty

`defined`

Příklad

```
#define ENG 1

#if defined(ENG) && ENG
    #define ERROR "error"
#else
    #define ERROR "chyba"
#endif
```

Vkládání souborů

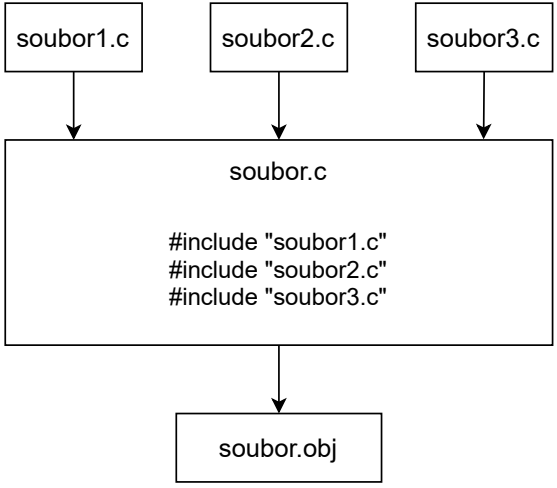
- `#include <nazev>`
- `#include "nazev"`

Další direktivy preprocesoru

- `#pragma`
- `#line` – nastavení hodnot maker `__LINE__` a `__FILE__`
- `#error` a `#warning`

Dělení do souborů

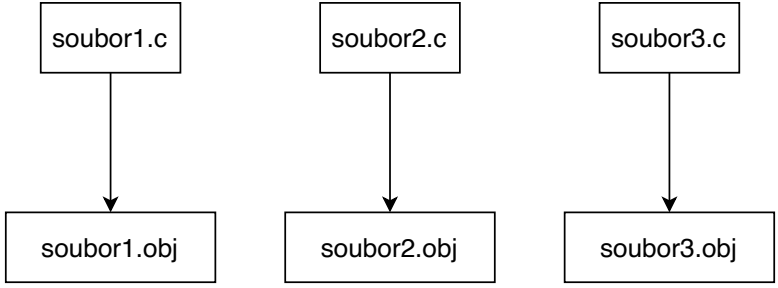
`#include`



sestavení: link soubor

Dělení do souborů

Oddělený překlad



sestavení: link soubor1, soubor2, soubor3

Dělení do souborů

Rozšíření platnosti globálních proměnných

Příklad (pomocny.c)

```c
int a;
extern int b;

int funkce1(){
    return a + b;
}
```

Příklad (hlavni.c)

```c
#include <stdio.h>

extern int a;
/* zde je nutne uvest cely
  funkcni prototyp */
extern int funkce1();
int b;

int main(){
    a = 3;
    b = 3;
    printf("%d\n", funkce1());
    return 1;
}
```

```
gcc -o program hlavni.c pomocny.c
```

Dělení do souborů

Statické globální funkce a proměnné

Příklad (pomocny.c)

```c
static int a;
extern int b;

static int funkce1(int x){
    return x;
}

int funkce2(){
    return funkce1(b) + a;
}
```

Příklad (hlavni.c)

```c
#include <stdio.h>

extern int a;
extern int funkce2();
int b;

int funkce1(){
    printf("%d\n", funkce2());
}

int main(){
    a = 3;
    b = 3;
    funkce1();
    return 1;
}
```

Dělení programu na menší části

- **zdrojový kód** – soubor s příponou `.c`
- **hlavičkový soubor** – soubor s příponou `.h`

„Tvé externí identifikátory budou rozlišitelné podle prvních šesti znaků, i když ti toto tvrdé omezení bude ztěžovat život a doba, kdy bylo nutné, se ti bude zdát nekonečně vzdálena. Jinak si budeš rvát vlasy a rozum tvůj tě opustí v onen soudný den, kdy zatoužíš přeložit svůj program na starém systému.“

(Henry Spencer)

Dělení programu na menší části

Doporučený obsah `.c` souboru

- dokumentační část – jméno souboru a verze, stručný popis modulu, jméno autora a případně datum
- vložení všech potřebných hlavičkových souborů
- deklarace externích proměnných a funkcí
- globální proměnné, které se budou sdílet s ostatními moduly
- definice symbolických konstant a maker s parametry, které se používají pouze v tomto modulu
- definice lokálních typů (`typedef`)
- definice statických globálních proměnných
- funkční prototypy lokálních funkcí
- funkce `main()` (pokud jí modul obsahuje)
- definice ostatních funkcí

Dělení programu na menší části

Doporučený obsah `.h` souboru

- dokumentační část
- podmíněný překlad proti opakovanému vkládání
- definice symbolických konstant a maker s parametry, které se budou používat i v jiných modulech
- definice globálních typů
- deklarace globálních proměnných příslušného modulu
- úplné funkční prototypy globálních funkcí

Dělení programu na menší části

Konkrétní příklad

Konkrétní příklad najdete ve skriptech!

Cvičení

1. Pro porovnání různého chování maker a funkcí naprogramujte funkci `fna2`, která bude vracet druhou mocninu. Co bude jejím výsledkem pro argument i++?
2. Upravte kód ze slidu 13 tak, aby rozeznával čtyři různé jazyky.
3. Napište makro `na_3(x)`, které bude počítat třetí mocninu. Proměnné `i` a `j` předem definujte. Vyzkoušejte ho na následujících výrazech.
   ```
   na_3(3)
   na_3(i)
   na_3(2 + 3)
   na_3(i * j + 2)
   ```
4. Definujte makro `je_velike(c)`, které vrátí `0` není-li znak velké písmeno a `1`, pokud je.
5. Definujete makro `cti_int(i)`, které čte z klávesnice celé číslo. Makro musí jít použít i ve výrazu
   ```
   if((j=cti_int(k)) == 0)
   ```
   Nápověda: možná budete potřebovat operátor čárky.

Cvičení

6 Následující kód upravte tak, aby se ve funkci `funkce()` hodnoty proměnných vypisovaly jen v případě, že je makro `VERBOSE` nastaveno na 1.

Příklad

```
#include <stdio.h>
int funkce(int a, int b){
    int i, j = 0;
    for(i = 0; i < a; i = i + b){
        printf("i = %d \n", i);
        if(i > j){
            j = j + i;
        }
        printf("j = %d \n", j);
    }
    return i + j;
}
int main(){
    printf("Vysledek: %d ", funkce(50,5));
    return 0;
}
```

Cvičení

7. Přeložte předchozí programy s použitím přepínače `-E` (jak už víme, při překladu zdrojového kódu se provede jen předzpracování pomocí preprocesoru). Podívejte se na výsledek.

Příklad

```
gcc program.c -E -o program.txt
```

8. Vytvořte dva soubory `soubor1.c` a `soubor2.c`. Oba budou sdílet proměnnou `a`. Proměnná je definovaná v `soubor1.c` a v `soubor2.c` se vypisuje.
9. Změňte předchozí program tak, aby proměnná `a` byla definovaná jako globální `static`. Vyzkoušejte chování programu.
10. Vytvořte program `funkce.c`, který bude obsahovat jen funkce s funkčními prototypy `float plus(float a, float b);` a `float minus(float a, float b);` počítající součet (rozdíl) dvou čísel. Dále vytvořte program `main.c`, který bude obsahovat pouze funkci `main()`, ve které budou volány funkce `plus()` a `minus()`.
    Pomocí `#include` vložte soubor `funkce.c` do souboru `main.c` a přeložte.

Cvičení

11. Vyzkoušejte funkci programu z předchozího cvičení, pokud nebude `funkce.c` do souboru `main.c` vkládaná, ale budou překládány samostatně.
    1. Soubory odděleně překládejte a v souboru `main.c` uveďte úplné funkční prototypy funkcí `plus()` a `minus()`.
    2. Vytvořte soubor `funkce.h`, do něhož vložte úplný funkční prototypy funkcí `plus()` a `minus()`. Pomocí `#include` zajistěte spojení souborů `funkce.c` a souboru `main.c`.