

Práce s preprocesorem

6. cvičení

Jiří Zacpal

KMI/ZP2 – Základy programování 2

Preprocesor

- zpracovává zdrojový text programu před použitím překladače
- nekontroluje syntaktickou správnost programu
- provádí pouze záměnu textů
(např. identifikátor konstanty za její hodnotu)
- vypouští ze zdrojového souboru komentáře
- umožňuje podmíněný překlad programu
- řádky pro zpracování preprocesorem začínají znakem „#“, za kterým by neměla být mezera

Makra bez parametru (konstanty)

1/2

- definice konstanty obecně:
`#define jméno hodnota`
- příklady:
`#define PI 3.14`
`#define AND &&`
`#define ERROR printf("Chyba v programu!");`
- každý výskyt jména konstanty v následujícím textu se nahradí hodnotou této konstanty
- nenahrazují se výskyty jmen uzavřené v uvozovkách (např. `printf("Logická spojka AND...");`)
- předefinování symbolické konstanty není možné přímo, lze ji ale zrušit a pak znova definovat s jinou hodnotou
`#undef PI`
`#define PI 3.1416`

Makra bez parametru (konstanty)

2/2

- jména symbolických konstant bývá zvykem psát velkými písmeny (případně použít podtržítko)
- pokud je hodnota konstanty delší než jeden řádek, musí být na konci řádku znak „\“, který se při rozvoji makra vynechá

```
#define DLOUHA_KONSTANTA 12345678.12345\  
67890123456
```

- vzhledem k tomu, že zpracování maker spočívá v pouhém dosazení jednoho textu místo druhého, je někdy nutné uzavřít výrazy do závorek

```
#define POSUN ('a' - 'A')
```

```
...
```

```
znak = znak - POSUN;
```

Makra s parametry 1/2

- pokud v programu často používáme velmi krátkou funkci, bývá výpočet značně neefektivní, protože „administrativa“ spojená s voláním funkce je delší než samotné provedení příkazů v těle funkce
- místo použití klasické funkce lze použít makra s parametry, které nevytváří žádnou „administrativu“ za běhu programu, naopak jejich nevýhodou je vznik delšího programu
- definice makra obecně:
`#define jméno(arg1, ..., argN) tělo_makra`
- příklad:
`#define na2(x) ((x) * (x))`
- mezi jménem makra a následující závorkou nesmí být mezera
- jména makra s parametry bývá zvykem psát malými písmeny, případně použít znak podtržítka

Makra s parametry 2/2

- je rozumné uzavřít celé tělo makra do závorek a také každý argument v těle makra uzavřít do závorek
`#define je_cislice(c) ((c) >= '0' && (c) <= '9')`
- typická chyba při vynechání závorek kolem argumentů
`#define na2(x) x*x`
...
`v=na2(f-g);` se přepíše na `v=f-g*f-g;`
- typická chyba při vynechání vnějších závorek
`#define cti(c) c=getchar()`
...
`if(cti(c)=='A') ...` se přepíše na
`if(c=getchar()=='A') ...`
- pokud se parametr v těle makra vyskytuje vícekrát, neměl by mít odpovídající argument ve volání makra vedlejší účinek
`if(je_cislice(x++)) ...`
- u maker s parametry **nelze** použít rekurzi

Podmíněný překlad (PP)

- používá se, pokud chceme při překladu dočasně vynechat některou část zdrojového souboru
- typické příklady:
 - vynechání ladících částí programu (např. pomocné výpisy proměnných) po jeho odladění
 - platformově závislé části zdrojového kódu (např. různé cesty k adresáři v různých operačních systémech)
 - „zakomentování“ větší části kódu, který již obsahuje komentáře (komentáře nemohou být v ANSI C vhnížděné)

PP řízený konstantním výrazem

- syntaxe obecně:

```
#if konstantní_výraz  
    část_1  
#else  
    část_2  
#endif
```
- pokud je hodnota výrazu *konstantní_výraz* rovna nule, překládá se *část_2*, jinak se překládá *část_1*
- části *#else* a *část_2* lze vynechat
- příklad:

```
#define WINDOWS 1  
#if WINDOWS  
    #define JMENO "C:\\\\Data\\\\input.txt"  
#else  
    #define JMENO "/data/input.txt"  
#endif
```


PP řízený definicí makra 1/2

- syntaxe obecně:

```
#ifdef jméno_makra
    část_1
#else
    část_2
#endif
```

- pokud je makro *jméno_makra* definováno, překládá se *část_1*, jinak se překládá *část_2*
- části `#else` a *část_2* lze vynechat

- příklad:

```
#define WINDOWS
#ifdef WINDOWS
    #define JMENO "C:\\Data\\input.txt"
#else
    #define JMENO "/data/input.txt"
#endif
```

PP řízený definicí makra 2/2

- k dispozici je také direktiva, která je „opakem“ předchozí

- syntaxe obecně:

```
#ifndef jméno_makra
    část_1
#else
    část_2
#endif
```

- pokud je makro *jméno_makra* definováno, překládá se *část_2*, jinak se překládá *část_1*
- části *#else* a *část_2* lze vynechat
- příklad:

```
#ifndef WINDOWS
    #define JMENO "/data/input.txt"
#else
    #define JMENO "C:\\Data\\input.txt"
#endif
```

Další možnosti PP

- `#ifdef` a `#ifndef` testují existenci definice pouze jednoho jména makra, pro konstrukci složitějších podmínek lze použít operátor `defined`
- ve složitějších podmínkách PP lze také použít direktivu `#elif`, která má stejný význam `else if` v podmínkovém příkazu
- direktiva `#error` slouží pro výpis chyb při zpracování preprocesorem

- příklad:

```
#if defined(WINDOWS) && defined(DEBUG)
    #define LADENI 1
#elif !defined(DEBUG)
    #error Chyba nelze debugovat!
#else
    #define LADENI 2
#endif
```

Direktiva include

- slouží pro přidání celého obsahu uvedeného souboru na místo direktivy ve zdrojového kódu
- soubory mohou být umístěny buď ve stejném adresáři jako soubor, do kterého se přidávají, nebo v systémovém adresáři
- přidání souboru v systémovém adresáři:
`#include <název_souboru>`
- přidání souboru ve stejném adresáři:
`#include "název_souboru"`

Úkol

Napište makro `je_cislice(zaklad, znak)` pro testování, zda je daný znak (určen argumentem `znak`) číslicí soustavy s daným základem (argument `zaklad`). Makro `je_cislice` by mělo korektně fungovat pro základy soustav od 2 do 36 a libovolné znaky. Pro číslice s hodnotou větší než 9 používejte pro jednoduchost pouze velká písmena anglické abecedy.

Příklad použití:

```
if (je_cislice(8, '8') != 0) printf("Ano\n"); else  
printf("Ne\n");  
if (je_cislice(10+6, '0'+4) != 0) printf("Ano\n");  
else printf("Ne\n");  
if (je_cislice(30, '@') != 0) printf("Ano\n"); else  
printf("Ne\n");
```

Příklad výstupu:

```
Ne  
Ano  
Ne
```

Úkol – řešení

```
#define je_cislice(zaklad, znak)  
    ((znak)>64&&(znak)<91?  
    ((znak)-55<(zaklad))  
    : ((znak)>47&&(znak)<58?  
    ((znak)-48<(zaklad)):0))
```

Bodovaný úkol

Napište makro `cti_int(i)` pro načtení hodnoty typu `int` do proměnné `i`. Výsledná hodnota výrazu v těle makra by také měla odpovídat načtenému číslu.

Příklad použití:

```
int j, k;  
printf("Zadejte cele cislo: ");  
if ((j = cti_int(k)) == 0)  
printf("nula\n"); else printf("%i  
%i\n", j, k);
```

Příklad výstupu:

```
Zadejte cele cislo: 1  
1 1
```

Povolené knihovny: `stdio.h`, `stdlib.h`