Práce s preprocesorem

1 Zpracování programu

Než začneme s novou látkou, tak si připomeneme, jak se program vlastně zpracovává.

Nejprve program napíšeme. K tomu využijeme nějaký **Editor**. Součástí překladače je **Preprocesor**, který předzpracovává (upravuje) zdrojový kód tak, aby měl překladač snadnější práci. Například zajišťuje vynechání komentářů, vkládání dalších souborů a rozvíjí makra. Výsledkem je textový soubor, který se dál předává **Překladači** (compileru). Překladač provádí překlad zdrojového kódu (předzpracovaný procesorem) do objektového souboru (jazyk relativních adres). Vzniká `.OBJ` soubor. **Relativní kód** je téměř hotový program. Slovo relativní znamená, že adresy proměnných nebo funkcí ještě nejsou známy a jsou v OBJ zapsány relativně. Vedlejší produkt překladače je tzv. protokol o překladu `.LIS`, ve kterém je uložena informace o chybách nalezených překladačem. **Linker** (sestavovací program) přidělí relativním adresám skutečné adresy a upraví všechny odkazy (najde skutečné adresy) na dosud neznámé identifikátory. Výsledkem linkeru je spustitelný soubor `.EXE` . Kromě dříve zmíněných pojmů je třeba zmínit i **Debugger** (ladící program), který slouží pro ladění (nalézání chyb), které nastávají za běhu programu.

Zatím jsme preprocesor využívali nevědomky (`#include`). Preprocesor má mnoho možností, které dávají jazyku C další sílu.

Preprocesor

- zpracovává zdrojový kód před překladačem
- nekontroluje syntaktickou správnost
- provádí záměnu textů
- odstraňuje z kódu komentáře
- připravuje podmíněný překlad

Řádky zpracovávané preprocesorem začínají `#` (přičemž před ani za `#` by neměla být mezera).

2 Makra

2.1 Makra bez parametru (konstanty)

Makra bez parametrů nazýváme **Symbolické konstanty**. Symbolické konstanty zbavují kód tzv. magických nepojmenovaných čísel.

Například v následujícím vzorečku na výpočet obvodu kružnice je jím číslo 3.14.

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

Tyto makra se definují na začátku programu. Preprocesor při úpravě kódu nahradí tyto konstanty skutečnou hodnotou (tomuto procesu se říká expanze, nebo také rozvinutí). Konvencí je, že se název píše velkými písmeny.

Syntaxe:

```
#define JMENO hodnota
```

Pozor! Na konec řádku se nepíše ;

Zde je uvedeno několik příkladů:

```
#define PI 3.14
#define AND &&
#define ERROR printf("Chyba programu");
```

Při zpracování je každý výskyt jména konstanty v následujícím textu zdrojového kódu nahrazen hodnotou této konstanty.

```
#define PI 3.14

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

Výjimkou jsou výskyty uzavřené v uvozovkách (části textového řetězce), viz následující příklad.

```c
#define JMENO "Marketa"

int main()
{
    // Vypise Moje jmeno je JMENO
    printf("Moje jmeno je JMENO");

    return 0;
}
```

Jakým způsobem bychom vypsali jméno? Zkuste se zamyslet, řešení najdete na konci textu.

Platnost definice konstanty je od definice až do konce souboru. Pokud je nutné změnit hodnotu konstanty, je nutné jí zrušit (`#undef JMENO`) a znovu definovat.

```c
#define JMENO "Marketa"

#undef JMENO

#define JMENO "TRNECKOVA"
```

Pokud je potřeba definici konstanty napsat na více řádků, přidáme na konec každého řádku znak \

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

Preprocesor tento znak vynechá a pokračuje ve zpracování hodnoty na následujícím řádku.

2.2 Makra s parametry (inline funkce)

Pokud v programu používáme funkci, která je tvořena velmi malým počtem příkazů, bývá výpočet značně neefektivní, protože „administrativa“ spojená s voláním funkce (předání parametrů funkci, skok do funkce, úschova návratové adresy, ...) je výpočetně náročnější, než provedení příkazů v těle funkce.

Místo klasické funkce lze použít makro s parametry, které nevytváří žádnou „administrativu“ za běhu programu.

Nevýhodou je vznik delšího (většího) programu, nemožnost použít rekurzi.

Konvence je, že se názvy takovýchto maker píšou malým písmem, jako klasické funkce.

```c
#define jmeno(arg1, arg2, ..., argn) telo_makra
```

`arg1, ..., argN` se chovají podobně jako argumenty funkcí.

Pozor! Mezi jménem a závorkou nesmí být mezera (bylo by to bráno jako konstanta).

Vzhledem k tomu, že při expanzi maker dochází pouze k nahrazení jednoho textu jiným, je poporučováno uzavřít celé tělo makra do závorek a stejně tak každý výskyt argumentů, viz následující příklad.

```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
```

Také se vyvarujte použití argumentů s vedlejším efektem. Podívejte na následující kód. Než ho zkusíte přeložit, zamyslete se, co bude výsledkem.

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

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

Pro porovnání různého chování maker a funkcí naprogramujte funkci `fna2`, která bude vracet druhou mocninu. Co bude jejím výsledkem pro argument i++?

2.3 Podmíněný překlad (dále PP)

PP používáme pro dočasné vynechání části kódu při kompilaci. Typicky se používá pro vynechání ladících částí programu, překlad platformově závislých částí zdrojového kódu či dočasné odstranění (zakomentování) větší části zdrojového kódu.

Syntaxe vypadá následovně.

```c
#if podminka1
    cast 1
#elif podminka2
```

```
    cast2

    ...

#else
    cast 3
#endif
```

Preprocesor vyhodnocuje podmínku, která následuje za `#if` . Pokud je tato podmínka vyhodnocena jako `false`, pak vše mezi `#if` a `#endif` je ze zdrojového kódu vypuštěno. Za podmínkou `#if` může být další podmínka `#elif` (může jich být více). Ta se vyhodnotí v případě, že předchozí podmínka nebyla splněna. Pokud nejsou splněny žádné podmínky, vyhodnotí se část kódu za `#else` (není povinný). Podmíněný překlad se ukončuje `#endif`.

Pozor! V podmínce musí být výrazy, které může vyhodnotit preprocesor (například v nich nelze používat hodnoty proměnných).

2.3.1 PP řízený konstantním výrazem

```
#if konstantni_vyraz
    cast 1
#else
    cast 2
#endif

/* NEBO */

#if konstantni_vyraz
    cast 1
#elif konstantni_vyraz2
    cast2
#else
    cast 3
#endif
```

Pokud je hodnota konstantního výrazu nenulová, překládá se část 1, jinak se překládá část 2.

Konkrétní příklad (Tento příklad se bude modifikovat ve cvičení 1):

```
#define ENG 1

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

K dispozici je i direktiva `#ifdef` a `#ifndef`.

`#ifdef JMENO` znamená, že kód následující za touto podmínkou se vykoná jen pokud je makro JMENO definováno (obdobně `#ifndef JMENO`, pokud JMENO není definováno).

```
#define ENG 1

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

`#ifdef` a `#ifndef` testují existenci jednoho makra. Pro složitější podmínky je potřeba použít operátor `defined` a logické spojky. Viz následující modifikace příkladu.

```
#define ENG 1

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

Na rozdíl od `#ifdef` a `#ifndef`, je možné použít složitější větvení pomocí `#elif`.

```
#define ENG 1

#if defined(ENG) && ENG
    #define ERROR "error"
#elif !defined(ENG)
    #define ENG 0
    #define ERROR "chyba"
#else
    #define ERROR "chyba"
#endif
```

3 Vkládání souborů

Bez dřívějšího vysvětlení jsme používali příkaz `#include <nazev>` , abychom mohli používat funkce z knihovny `nazev`.

Preprocesor tento příkaz nahradí obsahem souboru `nazev` (tento soubor je do kódu inkludován).

`#include` lze použít dvěma způsoby:

`#include <nazev>` (jak jsme používali doteď) a `#include ''nazev''` . Tyto dva příkazy se liší tím, kde se soubor `nazev` má hledat.

`#include ''nazev''` hledá soubor ve stejném adresáři, ve kterém je uložen „volající“ program. Používá se zejména pro vkládání souborů, které jsme vytvářeli sami.

`#include <nazev>` hledá v systémovém adresáři. Používá se pro vkládání standardních knihoven.

4 Cvičení

1. Upravte kód z příkladu výše, tak aby se rozeznávaly 4 různé jazyky pro chybovou hlášku.
2. Napište program, který sečte `N` prvních přirozených čísel a vypíše např. "Soucet prvnich 5-ti cisel je 15."N definujte jako symbolickou konstantu.
3. Napište makro `na_3(x)`, které bude počítat třetí mocninu. Vyzkoušejte ho na 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)
   ```

   Můžete použít například funkci getchar.

5 Řešení

Jak vypsat jméno:

```c
#define JMENO "Marketa"

int main()
{
    // Vypise Moje jmeno je JMENO
    printf("Moje jmeno je %s", JMENO);

    return 0;
}
```