Úvod do jazyka C

Tato kapitola je věnována představení jazyka C, jeho historie a také způsobu zpracování kódu. Dále zde budou zmíněny základní pojmy, které budeme v dalším textu používat.

Jazyk C

Jazyk C je nízkoúrovňový programovací jazyk, který je primárně určen pro systémové programování. Je to strukturovaný *imperativní* (procedurální) jazyk. To, že je imperativní, znamená, že výpočet je popsán posloupností příkazů a určuje přesný postup (algoritmus), jak danou úlohu řešit. *Strukturovaný* znamená, že skoky typu goto nahrazuje pomocí podmíněných cyklů[1] a jiných strukturovaných příkazů, které je možné do sebe vnořovat.
Jazyk C je slabě[2] staticky typovaný[3] jazyk s lexikálním rozsahem platnosti proměnných.
Jazyk C významně ovlivnil syntax některých moderních jazyků, jako je Java, JavaScript, C#, PHP a jiné. Pro mnoho úloh je efektivnější a rychlejší, než jiné jazyky.
Jazyk C byl navržen a implementován pod operačním systémem UNIX a téměř celý UNIX je psán v jazyce C. Přesto se jazyk C na OS UNIX nijak neváže a neváže se ani na jiný OS nebo počítač. Programy psané v jazyce C jsou snadno přenositelné mezi počítači a operačními systémy.

[1] Opakuj dokud platí podmínka.

[2] Slabý = dovoluje přiřadit hodnoty jednoho typu do jiného.

[3] Staticky typovaný = ověřuje typy výrazů, zda odpovídají svým kontextům.

Historie

První kniha o jazyce C, *The C Programming Language*,[4] vyšla v roce 1978. Verze jazyka popsána v této knize byla považována za jeho první standard. V literatuře se můžeme setkat s označením *K&R standard*.
V roce 1988 vyšel nový standard *ANSI C*, který z K&R standardu vycházel. Jeho součástí je i přesná specifikace knihoven, kterou každá implementace ANSI C musí obsahovat. Naprostá většina dnešních překladačů této normě vyhovuje. Proto se této normě budeme v tomto

[4] Kernighan, B. W. and Ritchie, D. M. The C Programming Language. Prentice-Hall, Inc. (1978).

kurzu věnovat.[5] Výhodou používání této normy je, že program napsaný v tomto standardu s použitím pouze standardních knihoven je téměř 100% přenositelný na libovolný počítač.[6]
V současné době existuje rozšiřující standard *C99*.[7] Jeho některé prvky[8] nemusí všechny překladače podporovat. Na tyto prvky budeme dále upozorňovat. Novinky jsou spíše kosmetického rázu. Některá vylepšení představená v dalších normách, například *C11*, *C1X*,[9] se neujala a nejsou plně podporována.

[5] Konkrétně verzi ISO/ANSI C x3.159-189

[6] Pod libovolný operační systém.

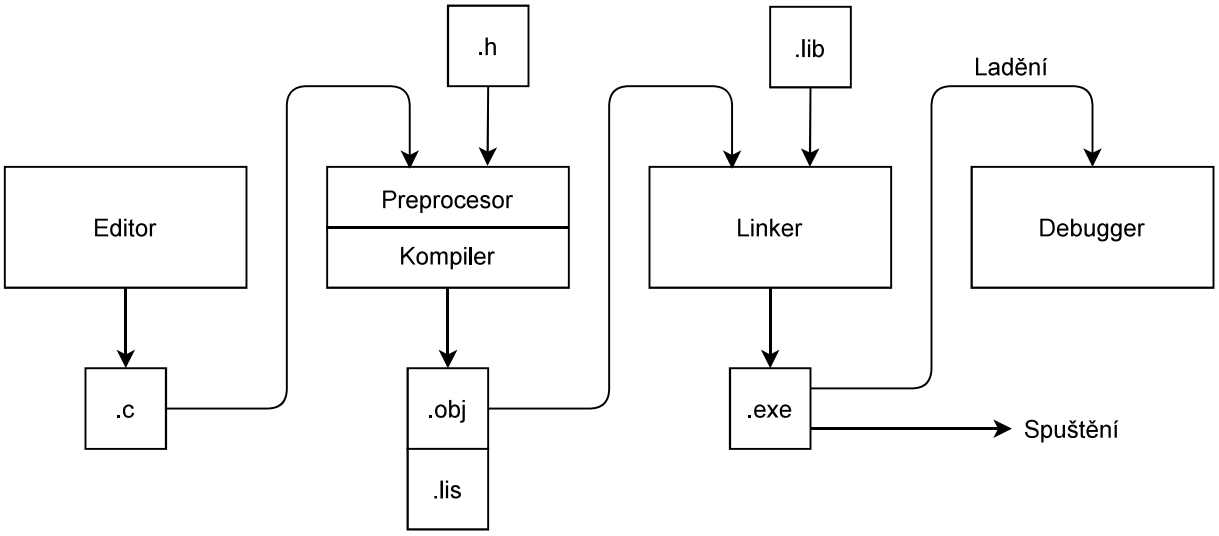
[7] Formálně ISO/IEC 9899:1999

[8] Například inline funkce, možnost definovat proměnné kdekoliv, nové datové typy, dynamická pole aj.

[9] ISO/IEC 9899:2011

Způsob zpracování programu

Způsob zpracování programu zapsaného v jazyce C je schématicky zobrazen na obrázku níže.



Nejprve je pomocí *editoru* vytvořen zdrojový kód, který se ukládá s příponou `.c`. Ten je poté zpracován pomocí *preprocesoru*, který je součástí *kompileru* (kompilátoru, překladače). Preprocesor, jak už název napovídá, předzpracovává zdrojový kód[10] tak, aby měl překladač jednodušší práci. Výsledkem je textový soubor, který je přímo předáván kompileru. Kompiler provádí překlad zdrojového souboru předzpracovaného preprocesorem do relativního (objektového) kódu počítače (vzniká soubor s příponou `.obj`). *Relativní kód*, nebo také jazyk relativních adres, je téměř hotový program. Adresy proměnných a funkcí nejsou známé,[11] jsou v souboru zapsány relativně. Vedlejším produktem je soubor s příponou `.lis`, ve kterém je uložena informace o chybách nalezených kompilerem.[12] Tento soubor se dále nepoužívá a generuje se jen na přání uživatele. *Linker* (sestavovací program) přidělí relativním adresám absolutní adresy a provede všechny odkazy na dosud neznáme identifikátory.[13] Výsledkem je spustitelný soubor s příponou `.exe` (v textu předpokládáme pod operačním systémem Windows). *Debugger* (česky

[10] Např. vynechává komentáře, zajišťuje správné vložení hlavičkových souborů (souborů s příponou `.h`), stará se o rozvoj maker a jiné.

[11] Například jsou uloženy v knihovně.

[12] Například syntaktické chyby.

[13] Například na volané knihovní funkce uložené v souborech s příponou `.lib`.

„odvšivovač“) je program sloužící k hledání chyb (ladění), které nastávají za běhu programu.

První program

Klasický program, který vypíše na obrazovku „ahoj svete“, zapsaný v jazyce C vypadá následovně.

```c
#include <stdio.h>

int main()
{
    /*program vypise ahoj svete*/
    printf("ahoj svete");
    return 0;
}
```

Program obsahuje funkci `main()`. Obecně můžeme pojmenovávat funkce téměř libovolně,[14] ale všechny programy musí obsahovat právě jednu funkci `main()`.[15] Vykonávání programu začíná na začátku funkce `main()`. Ve funkci `main()` jsou většinou volány další funkce, ať už uživatelsky definované nebo dostupné ve formě ve formě knihoven.
O výpis textu na obrazovku se stará funkce `printf()`.[16] Kód neobsahuje její definici, abychom jí mohli použít, je potřeba informaci o této funkci do kódu vložit. `printf()` funkce je součástí standardní knihovny pro vstup a výstup `stdio` (Standard Input/Output) a o její vložení do kódu se stará příkaz

[14] Více se pojmenovávání proměnných a funkcí budeme věnovat v části Štábní kultura.

[15] Jednu funkci `main()` musí obsahovat i v případě, kdy se program skládá z více souborů (modulů).

[16] Více si o této funkci řekneme později.

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

Jak již víme z dřívějšího studia, funkcím je možné předávat argumenty. V jazyce C se argumenty píší do kulaté závorky hned za název funkce. V našem případě je funkce `main()` definována tak, že neočekává žádné vstupní argumenty. `int` před názvem funkce udává typ návratové hodnoty.[17]
Příkazy funkce jsou uzavřeny v `{` a `}` (příkazům uzavřeným ve složených závorkách říkáme *blok*). Jednotlivé příkazy jsou odděleny `;`.
Funkce `main()` mimo jiné obsahuje příkaz

[17] Funkcím a datovým typům se budeme věnovat později.

```c
printf("ahoj svete");
```

Příkazem je volána funkce `printf()`, které předáváme argument ”`Ahoj svete`”. Funkce je ukončena příkazem `return`, o jehož významu si řekneme v následující kapitole.

Kód je možné doplnit o *komentáře*. Komentáře se píší do bloku ohraničeného `/*` a `*/`. Tyto komentáře mohou být víceřádkové a mohou se objevit všude, kde se v kódu vyskytuje bílý znak.[18] V komentáři se mohou vyskytovat libovolné znaky. ANSI C standard nepovoluje vnořené komentáře.[19]
Komentáře jsou důležitou součástí programu, slouží k jeho zpřehlednění, je tedy dobré je používat před každou logicky ucelenou částí. Komentář by měl obsahovat jen důležité informace týkající se kódu.

[18] Bílým znakem rozumíme mezery, tabulátory, nové řádky a podobně.

[19] `/* /* vnoreny */ komentar */`

Nový standard ISO
Nový standard umožňuje *jednořádkové komentáře*. Komentář začíná `//` a končí s koncem řádku.

Štábní kultura

Než si vyzkoušíme napsat a přeložit první program, je dobré si říci něco o pojmenovávací konvenci, která se v jazyce C používá.
Při pojmenovávání souboru obsahujícího kód v jazyce C se snažíme vyhnout nejednoznačným názvům. Název by měl odpovídat tomu, co daný program dělá. Pokud se program skládá z více souborů (modulů),[20] použijeme několik prvních znaků názvu pro identifikaci projektu a zbytek pro název modulu. Například `pr1_vstup.c`, `pr1_vystup.c`, ...
Každá proměnná a funkce má svůj název, kterému říkáme *identifikátor*. Jazyk C je *case sensitive*, což znamená, že rozlišuje malá a velká písmena. `id`, `Id`, `ID` označují tři různé identifikátory. Identifikátor je v jazyce C kombinace malých a velkých písmen, číslic a znaku `_` (podtržítko). Identifikátor nemůže začínat číslicí a jako identifikátor není možné použít *klíčová slova* jazyka.[21] Názvy začínající podtržítkem se používají pro systémové identifikátory. Pro přehlednost se nedoporučuje podtržítko používat na konci názvu. Délka identifikátoru není omezená, ale ANSI C rozeznává pouze prvních 31 znaků (další bere jako nevýznamné).
Obecně se doporučuje používat v názvech identifikátorů malá písmena a využívat podtržítka (této konvenci se říká *snake konvence*). Znak `_` je použit pro oddělení slov (pokud jich je více). Například `nazev_promenne` nebo `ukazatel_na_cislo`. Nedoporučuje se používat podobné názvy[22] nebo používat stejné identifikátory lišící se pouze velikostí písma.[23] Názvy by měly naznačovat, co funkce dělá, případně jakou informaci proměnná obsahuje.[24] Běžně se v jazyce C používají následující významové identifikátory:

- `i`, `j`, `k` – indexy
- `c`, `ch` – znaky
- `m`, `n` – čítače
- `f`, `r` – reálná čísla
- `p_` – začátek ukazatele (pointeru)

[20] Rozdělení programu do více souborů se budeme věnovat později.

[21] Slova, která mají v jazyce speciální význam, například `while`, `if`, `for`, ... Klíčová slova jsou psána malými písmeny, jinak jsou brána jako identifikátory. `While` je identifikátor, zatímco `while` je klíčové slovo.

[22] Například `pr1` a `prl`.

[23] Například `Id` a `id`.

[24] Z názvu `plat` je zřejmé, co identifikátor uchovává za informaci, na rozdíl od nic neříkajícího názvu `promenna_1`.

- s – řetězce

Cvičení

Úkol 1

1. Nainstalujte si překladač jazyka C. V ukázkách textu bude použit *gcc* překladač.
2. Pomocí libovolného textového editoru vytvořte soubor `ahoj.c`. Soubor bude obsahovat následující kód:

```c
#include <stdio.h>

int main()
{
    /*program vypise ahoj svete*/
    printf("ahoj svete");
    return 0;
}
```

3. Otevřete příkazový řádek (případně terminál), přesuňte se do složky, kde máte uložený soubor `ahoj.c` a pomocí následujícího příkazu jej přeložte:

```
gcc ahoj.c -o ahoj
```

Pomocí přepínače `-o` specifikujeme, jak se má přeložený soubor jmenovat.[25] Ve složce vznikl soubor ahoj.exe, který je možné spustit. Po spuštění se do příkazového řádku vypíše text „ahoj svete“. Viz následující obrázek.

Ve většině systémů je možné tyto dva kroky sloučit do jednoho příkazu s použitím `&&`

[25] Pomocí příkazu `gcc --help` zobrazíte nápovědu k příkazu `gcc`.

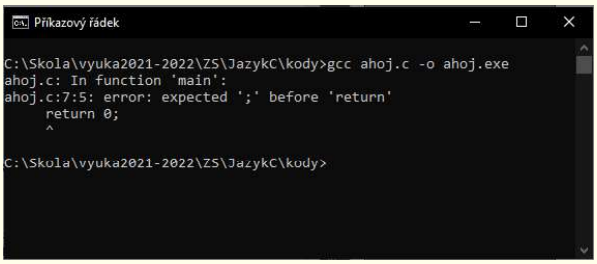
```
gcc ahoj.c -o ahoj && ahoj
```

4. Použitím přepínače `-E` při překladu zdrojového kódu se provede jen předzpracování pomocí preprocesoru. Vyzkoušejte následující příkaz a podívejte se na vygenerovaný soubor `ahoj.txt`.

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

5. Vyzkoušejte i jiné přepínače, například `-S`, nebo `-c`.

Úkol 2
Vyzkoušejte smazat středník za příkazem `printf()` a kód přeložte. V příkazovém řádku by se měla objevit chyba při překladu, viz následující obrázek.



Při použití složeného příkazu pro přeložení a spuštění

```
gcc ahoj.c -o ahoj && ahoj
```

se v tomto případě program nespustí, pouze se vypíše seznam chyb.

Úkol 3
Nainstalujte si libovolné vývojové prostředí[26] a vyzkoušejte předchozí kód zkompilovat pomocí něj. [27]

Úkol 4
Upravte předchozí kód tak, aby vypsal jiný text.

[26] Vývojové prostředí, tzv. *IDE* (Integrated Development Environment) obsahuje editor zdrojového kódu, kompilátor (případně interpret) a většinou také debugger.

[27] V textu budeme používat CodeBlock `http://www.codeblocks.org/`. Dále je možné využívat například Visual Studio (v tom případě je třeba vytvářet prázdný projekt).