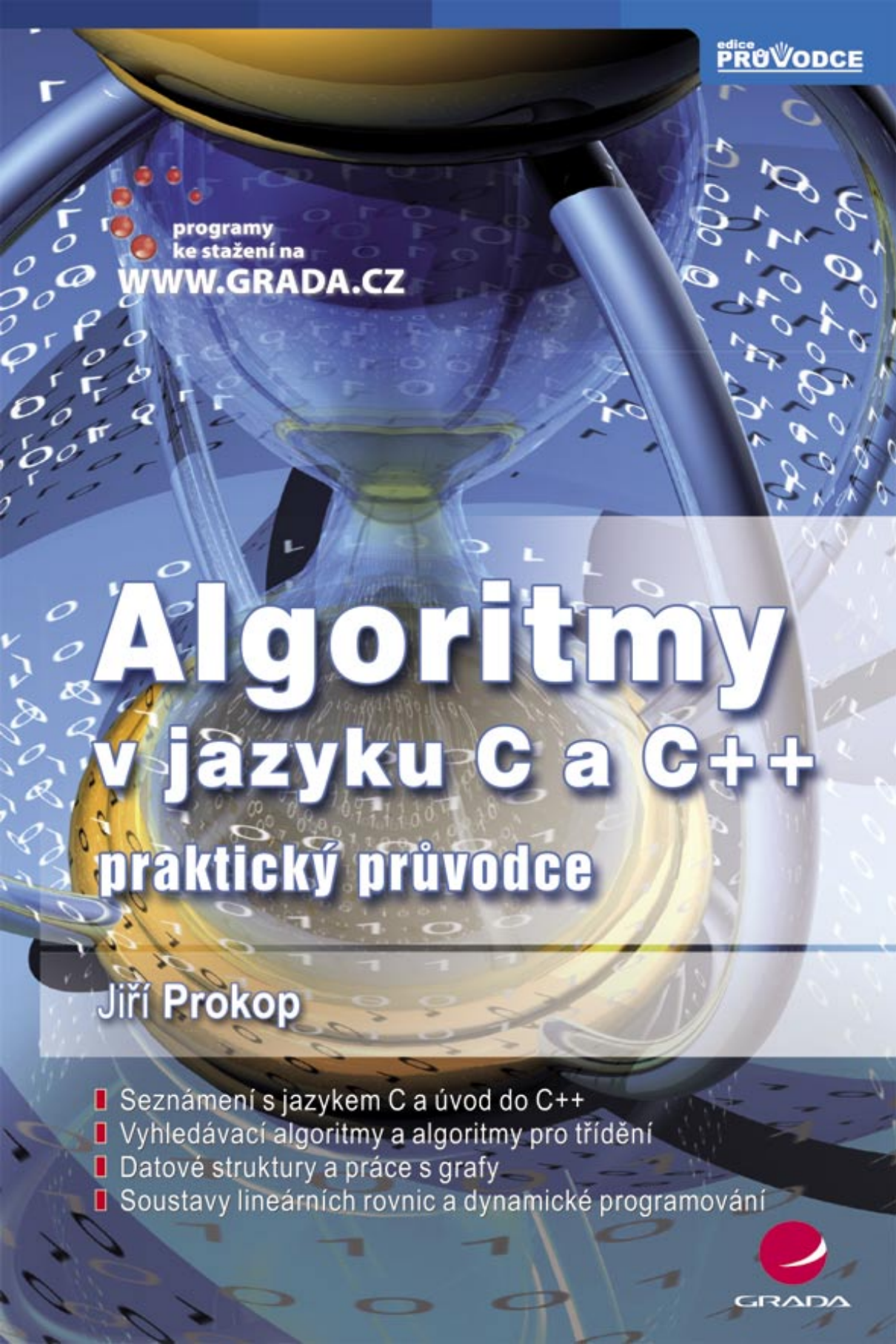
edice
PROVODCE
programy
ke stažení na
WWW.GRADA.CZ
Algoritmy
v jazyku C a C++
praktický průvodce
Jiří Prokop
Seznámení s jazykem C a úvod do C++
Vyhledávací algoritmy a algoritmy pro třídění
Datové struktury a práce s grafy
Soustavy lineárních rovnic a dynamické programování
GRADA

Upozornění pro čtenáře a uživatele této knihy

Algoritmy v jazyku C a C++

praktický průvodce

Jiří Prokop

Vydala Grada Publishing, a.s.
U Průhonu 22, Praha 7
jako svou 3473. publikaci

Odpovědný redaktor Zuzana Malečková
Návrh vnitřního layoutu Miroslav Lochman
Sazba Eva Grillová
Návrh a grafická úprava obálky Vojtěch Kočí
Počet stran 160
První vydání, Praha 2009

Vytiskly Tiskárny Havlíčkův Brod, a.s.
Husova ulice 1881, Havlíčkův Brod

ISBN 978-80-247-2751-6 (tištěná verze)
ISBN 978-80-247-6622-5 (elektronická verze ve formátu PDF)

Obsah

Úvod

V r. 2002, kdy jsem začal na Gymnáziu Christiana Dopplera vést seminář s názvem „Programování v jazyku C“, neexistovala na našem knižním trhu učebnice, která by se věnovala algoritmům a používala jazyk C. Algoritmy byly po řadu let prezentovány téměř výlučně v jazyku Pascal, např. [Wir89] a [Top95]. Musel jsem tedy během šesti let algoritmy pro účely výuky naprogramovat, a tak vznikl základ této knihy.

Kniha se nesnaží být učebnicí jazyka C, i když může být k užitku všem, kteří jazyk právě studují. Dobrých učebnic jazyka je dostatek, doporučit lze např. [Her04] nebo [Ka01], pro C++ [Vi02], [Vi97]. Jestliže jsem přesto zařadil do knihy alespoň stručný přehled jazyka C a také úvod do C++, je to proto, aby čtenář měl při studiu knihy vše potřebné pro porozumění zdrojovým textům algoritmů a nemusel hledat informace jinde.

Kdo je s jazykem C seznámen do té míry, že chápe nejdůležitější operátory, výrazy a přiřazení, příkazy pro řízení programu, příkazy vstupu a výstupu, funkce a vedle jednoduchých datových typů ještě pole, stačí mu to už ke studiu jednoduchých algoritmů. Takový přehled jazyka obsahuje právě první kapitola. Poté lze studovat kapitolu druhou, věnovanou rekurzi, a třetí, která se zabývá třídicími algoritmy. Teprve pro studium datových struktur je v kapitole 4 nutno rozšířit zatímní podmnožinu jazyka o struktury a dynamické přidělování paměti. Tyto znalosti jsou pak potřebné i pro pochopení algoritmů na grafech

a pro vyhledávání pomocí binárních stromů. Stromy se využívají také k reprezentaci aritmetických výrazů a pro počítačové řešení hlavolamů a her. Popsaná podmnožina jazyka je v těchto kapitolách dále rozšiřována podle potřeby. Algoritmy z kapitol 1 až 8 jsou napsány v jazyku C. Teprve 9. kapitola je úvodním popisem C++ a algoritmy v následujících kapitolách jsou v C++.

Z tohoto stručného průvodce obsahem knihy vyplývá samozřejmě doporučení studovat jednotlivé kapitoly postupně a bez přeskakování, protože v každé kapitole se počítá se znalostmi, které si čtenář přináší z kapitol předchozích. Dalším doporučením je studium aktivní. Usnadňuji ho tím, že všechny algoritmy rozdělené podle kapitol knihy lze najít na webových stránkách www.grada.cz. Zdrojové texty tedy nemusí nikdo pracně vkládat, čtenář může provádět v programech úpravy, mnohde k tomu zdrojový text přímo vybízí tím, že části zdrojového textu jsou „ukryty“ v komentářích. Často lze algoritmus snáze pochopit, zobrazíme-li si některé mezivýsledky. Aktivní způsob studia je mimo jiné určitě mnohem zajímavější. Algoritmy jsou ověřeny s použitím kompilátoru Dev C++ a kompilátoru Microsoft Visual C++. Kdyby čtenáři měli ke knize jakékoli připomínky, mohou je sdělit na e-mailovou adresu Jiri_Prokop@yahoo.com. Přeji svým čtenářům mnoho úspěchů ve studiu.

1.

Jazyk C

1.1 Stručný přehled jazyka C

Jazyk C rozlišuje velká a malá písmena. „Prog“, „prog“ a „PROG“ jsou tedy tři různé identifikátory. Identifikátory sestávají z písmen, číslic a podtržítka, číslice nesmí být na prvním místě. Pro oddělování klíčových slov, identifikátorů a konstant slouží oddělovače (tzv. „bílé znaky“). Všude tam, kde mohou být oddělovače, může být komentář:

```
/* toto je komentář */
```

Struktura programu: direktivy preprocesoru, deklarace, definice, funkce. V každém programu je právě jedna funkce hlavní (main), která se začne po spuštění programu vykonávat.

1.1.1 Deklarace

Deklarace jsou povinné. Deklaraci jednoduché proměnné tvoří specifikátor typu a jméno (identifikátor proměnné)

```
int a;                /* deklarace celočíselné proměnné a */
int b=1;              /* definice proměnné b */
```

Podle umístění dělíme deklarace na globální (na začátku programu) a lokální (v těle funkce). Lokální proměnné nejsou implicitně inicializovány a obsahují náhodné hodnoty.

Specifikátory typu pro celá čísla: `int`, `char`, `short int` (nebo jen `short`), `long int` (nebo jen `long`).

Každý z nich může být `signed` (se znaménkem) nebo `unsigned` (bez znaménka), implicitně je `signed`.

Specifikátory typu pro racionální proměnné: `float` (32 bytů), `double` (64), `long double` (80).

U konstant je typ dán způsobem zápisu. Pomocí klíčového slova const můžeme deklarovat konstantní proměnnou, jejíž obsah nelze později měnit:

```
const float pi=3.14159;
```

1.1.2 Výrazy a přiřazení

Výrazy jsou v jazyce C tvořeny posloupností operandů a operátorů. Operátory dělíme podle arity (počet operandů) na unární, binární a ternární, podle funkce na aritmetické: `+`, `-`, `*`, `/`, `%` pro zbytek po dělení (operátor `/` má význam reálného nebo celočíselného dělení podle typů operandů), relační: `>`, `<`, `>=`, `<=`, `==` (rovnost), `!=` (nerovnost), logické: `||` (log.součet), `&&` (log.součin), `!` (negace). Jazyk C nezná logický typ, nulová hodnota představuje `true`, nulová `false`.

Podmíněný operátor ? (jediný ternární operátor)

```
x=(a<b) ? a:b;
```

má stejný význam jako

```
if (a<b)       x=a; else x=b;
```

Obecně

```
v1 ? v2 : v3
```

`v1` je výraz, jehož hodnota je po vyhodnocení považována za logickou. Je-li `true`, vyhodnotí se výraz `v2` a vrátí se jeho hodnota, je-li `false`, pak se vyhodnotí `v3` a vrátí se jeho hodnota. `v2` a `v3` jsou jednoduché výrazy.

Operátory přiřazení

```
a=a+b;

a+=b;          /* má význam a=a+b; */
```

Na místě + může být -, *, /, %, & a další, o nichž zatím nebyla řeč.

Operátory inkrementace a dekrementace

```
a++;           /* postfixová verze */

--a;           /* prefixová verze */
```

Příklad:

```
a=10;
x=++a;  /* x bude mít hodnotu 11, a taky */
y=a--;  /* y=11, a=10 */
```

Unární operátory

Adresní operátor `&`, operátor dereference `*`, unární `+`, unární `-`, logická negace `!` a prefixová inkrementace `++` a dekrementace `--`. K postfixovým operátorům patří operátor přístupu k prvkům pole `[ ]`, operátor volání funkce `( )`, postfixová inkrementace `++` a dekrementace `--` a operátory přístupu ke členům struktury, jimž se budu věnovat později.

Operátor přetypování ukáži na příkladu (`i1` a `i2` jsou celočíselné proměnné, ale chci reálné dělení):

```
f=(float) i1/i2;
```

Operátor sizeof pro zjištění velikosti: argumentem operátoru může být jak název typu, tak identifikátor proměnné.

1.1.3 Priorita a asociativita operátorů

Priorita	*Operátory*	*Vyhodnocuje se*
1	`( )` `[ ]` `->` (funkce, index, přístup k prvku struktury)	zleva doprava
2	`!` `-` `++` `--` `+` (typ) `*` `&` `sizeof` (unární operátory)	zprava doleva
3	`*` `/` `%` (multiplikativní operátory)	zleva doprava
4	`+` `-` (aditivní operátory)	zleva doprava
5	`<<` `>>` (operátory posunů)	zleva doprava
6	`<` `<=` `>` `>=` (relační operátory)	zleva doprava
7	`==` `!=` (rovnost, nerovnost)	zleva doprava
8	`&` (operátor bitového součinu)	zleva doprava
9	`^` (exklusivní nebo)	zleva doprava
10	`|` (operátor bitového součtu)	zleva doprava
11	`&&` (operátor logického součinu)	zleva doprava
12	`||` (operátor logického součtu)	zleva doprava
13	`?:` (ternární podmínkový operátor)	zprava doleva
14	`=` `+=` `-=` `*=` `/=` `%=` `>=` `&=` `|=` `^=` (oper. přiřazení)	zprava doleva
15	`,` (operátor čárky)	zleva doprava

1.1.4 Příkazy a bloky

Napíšeme-li za výraz středník, stává se z něj příkaz, jako je tomu v následujících příkladech:

```
float x,y,z;
x=0;
a++;
x=y=z;
y=z=(f(x)+3);  /* k hodnotě vrácené funkcí f je přičtena hodnota 3 */
               /* součet je přiřazen jak proměnné z, tak y */
```

Příkazy v jazyce C můžeme sdružovat do tzv. bloků nebo složených příkazů. Blok může obsahovat deklarace proměnných na svém počátku a dále pak jednotlivé příkazy. Začátek a konec bloku je vymezen složenými závorkami.

Složené příkazy používáme tam, kde smí být použit pouze jeden příkaz, ale potřebujeme jich více. Za uzavírací složenou závorkou se nepíše středník.

Příkaz if

má dvě podoby:

```
if (výraz) příkaz
if výraz příkaz1 else příkaz2;
```

Složitější rozhodovací postup můžeme realizovat konstrukcí `if else if`.

Každé `else` se váže vždy k nejbližšímu předchozímu `if`.

Příkaz switch a break

```
switch(výraz)
{
       case konst_výraz1:
       /* příkazy, které se provedou, když výraz = výraz1 */
              break;
       case konst_výraz2:
       /* příkazy, které se provedou, když výraz = výraz2 */
              ….
              break;
       default:       /* příkazy, které se provedou, není-li výraz
                      roven žádnému z předchozích konstantních výrazů */
}
```

Příkaz `break` říká, že tok programu nemá pokračovat následujícím řádkem, nýbrž prvním příkazem za uzavírající složenou závorkou příkazu `case`.

V těle příkazu switch budou provedeny všechny vnořené příkazy počínaje tím, na který bylo předáno řízení, až do konce bloku (pokud některý z příkazů nezpůsobí něco jiného – např. `break`). Tím se `switch` značně liší od pascalského `case`.

Příkaz while

```
while (výraz) příkaz;
```

Výraz za `while` představuje podmínku pro opakování příkazu. Není-li podmínka splněna už na začátku, nemusí se příkaz provést ani jednou. Je-li splněna, příkaz se provede a po jeho provedení se znovu testuje podmínka pro opakování cyklu.

Příkaz do-while

Zajistí aspoň jedno provedení těla cyklu, protože podmínka opakování se testuje na konci cyklu.

```
do příkaz while (výraz);
```

Příkaz for

```
for (inicializační výraz;podmíněný výraz;opakovaný výraz) příkaz
```

je ekvivalentní zápisu:

```
inicializační výraz;
while (podmíněný výraz)
{
        příkaz
        opakovaný výraz
}
```

Inicializační výraz může být vypuštěn, zůstane po něm však středník. Stejně může být vynechán i podmíněný výraz a opakovaný výraz. Příkaz `continue` je možno použít ve spolupráci se všemi uvedenými typy cyklů. Ukončí právě prováděný průchod cyklem a pokračuje novým průchodem. Podobně i příkaz `break` může být použit ve všech typech cyklů k jejich ukončení.

Příkaz goto a návěští

Příkaz `goto` přenese běh programu na místo označené návěštím (identifikátor ukončený dvojtečkou). Jsou situace, kdy může být užitečný, např. chceme-li vyskočit z vnitřního cyklu z více vnořených cyklů.

Prázdný příkaz

```
;
```

Lze ho použít všude tam, kde je prázdné tělo.

1.1.5 Preprocesor

Preprocesor zpracuje zdrojový text programu před překladačem, vypustí komentáře, provede záměnu textů, např. identifikátorů konstant za odpovídající číselné hodnoty a vloží texty ze specifikovaných souborů. Příkazy pro preprocesor začínají znakem # a nejsou ukončeny středníkem. Nejdůležitějšími příkazy jsou `#define` a `#include`.

```
#define ID hodnota
```

říká, že preprocesor nahradí všude v textu identifikátor ID specifikovanou hodnotou, např.

```
#define PI 3.14159
#include <stdio.h>
```

znamená příkaz vložit do zdrojového textu funkce vstupu a výstupu ze systémového adresáře.

```
#include "filename"
```

znamená, že preprocesor vloží text ze specifikovaného souboru v adresáři uživatele.

Některé standardní knihovny:

`stdio.h`	funkce pro vstup a výstup
`stdlib.h`	obecně užitečné funkce
`string.h`	práce s řetězci
`math.h`	matematické funkce v přesnosti double
`time.h`	práce s datem a časem

1.1.6 Funkce

Každá funkce musí mít definici a

- má určeno jméno, pomocí kterého se volá

- může mít parametry, v nichž předáme data, na kterých se budou vykonávat operace

- může mít návratovou hodnotu poskytující výsledek

- má tělo složené z příkazů, které po svém vyvolání vykoná. Příkazy jsou uzavřeny ve složených závorkách `{}`

Příkaz `return výraz;` vypočte hodnotu výraz, přiřadí ji jako návratovou hodnotu funkce a funkci ukončí.

Příklad:

```
int max(int a, int b)  /* hlavička */
{
        if (a>b)        return a;
        return b;
}
```

Nevrací-li funkce žádnou hodnotu, píšeme v místě typu návratové hodnoty `void`. Nepředáváme-li data, uvádíme jen kulaté závorky nebo `void`. Neznáme-li definici funkce a přesto ji chceme použít, musíme mít deklaraci funkce (prototyp), která určuje jméno funkce, paměťovou třídu a typy jejích parametrů. Na rozdíl od definice funkce již neobsahuje tělo a je vždy ukončena středníkem.

```
int max(int a, int b);
```

nebo jen

```
int max(int,int);
```

Pokud neuvedeme paměťovou třídu, je automaticky přiřazena třída `extern`. Je-li funkce definována v paměťové třídě `extern` (explicitně nebo implicitně), můžeme definici funkce umístit do jiného zdrojového souboru. Funkce je společná pro všechny moduly, ze kterých se výsledný program skládá a může být v libovolném modulu volána. Je-li deklarována ve třídě `static`, musí její definice následovat ve stejné překladové jednotce a je dostupná pouze v jednotce, ve které je deklarována a definována.

Volání funkcí:

```
výraz (seznam skutečných parametrů);
```

Nemá-li funkce žádné parametry, musíme napsat (). Parametry se vždy předávají hodnotou, jsou tedy následně překopírovány do formálních parametrů funkce. Rekurzivní funkce jsou v C dovoleny.

1.1.7 Vstup a výstup

Standardní vstup a výstup: `stdin`, `stdout`

Standardní vstup a výstup znaků

```
int getchar(void);            /* načte 1 znak */
int putchar(int znak);        /* výstup 1 znaku */
```

Pro načtení a výstup celého řádku znaků

```
char *gets(char *radek);
int puts(const char *radek);
```

Funkce `gets` načte znaky ze standardního vstupu, dokud není přechod na nový řádek. Ten už není do pole zapsán. Návratovou hodnotou je ukazatel předaný funkci jako parametr. Když došlo k nějaké chybě, má hodnotu `NULL`. Na řádku nesmíme zadat více znaků než je velikost pole. Funkce `puts` vypíše 1 řádek textu, za který automaticky přidá přechod na nový řádek. Řetězec samotný nemusí tento znak obsahovat. V případě, že výstup dopadl dobře, vrátí funkce nezápornou hodnotu, jinak `EOF`.

Formátovaný vstup a výstup

Funkce `printf` a `scanf` s následujícími deklaracemi:

```
int printf(const char *format,…);
int scanf(const char *format,…);
```

Obě funkce mají proměnný počet parametrů, který je určen prvním parametrem – formátovacím řetězcem. Formátovací řetězec funkce `printf` může obsahovat dva typy informací. Jednak jde o běžné znaky, které budou vytištěny, dále pak speciální formátovací sekvence znaků začínající % (má-li být % jako obyčejný znak, musím je zdvojit). K tisknutelným znakům patří také `escape` sekvence, např. `\n`.

`scanf` se liší tím, že formátovací řetězec smí obsahovat jen formátovací sekvence, a tím, že druhým a dalším parametrem je vždy ukazatel na proměnnou.

Formátovací sekvence (printf)

```
%[příznak] [šířka] [přesnost] [F] [N] [h] [l] [L] typ
```

typ

d,i	znaménkové decimální číslo typu int
o	neznam. oktalové číslo typu int
u	neznam. decimální číslo typu int
x,X	neznam. hexadecimální číslo typu int, pro x tištěno a, b, c, d, e, f, pro X pak tištěno A, B, C
f	znam. racionální číslo formátu [-] dddd.dddd
e,E	znam. racionální číslo v exp. formátu [-d]d.ddde[+|-]ddd.
g,G	znam. racionální číslo ve formátu bez exponentu nebo s exponentem (v závislosti na velikosti čísla)
c	jednoduchý znak
s	ukazatel na pole znaků ukončené nulovým znakem
p	tiskne argument jako ukazatel
n	ukazatel na číslo typu int, do kterého se uloží počet vytištěných znaků

příznak

-	výstup zarovnán zleva a doplněn zprava mezerami
+	u čísel vždy znaménko
mezera	kladné číslo mezera, záporné minus
#	závisí na typu

šířka

n	alespoň n znaků se vytiskne doplněno mezerami
0n	je vytištěno alespoň n znaků doplněných zleva nulami
*	šířka dána následujícím parametrem

přesnost

(nic)	je různá podle části typ
.0	stand.
.n	n des. míst
*	přesnost dána následujícím parametrem
h	argument funkce chápán jako short int – pouze pro d, i, o, u, x, X