# IB016 – 1. velký úkol – Argumenty nástroje `find`

### Termíny

- Neděle **17. dubna** 2022 23.59 – implementační část
- Neděle **24. dubna** 2022 23.59 – peer review

V tomto úkole si zkusíme napsat o něco rozsáhlejší kus kódu s využitím knihovny Parsec. Budeme zpracovávat argumenty klasického unixového nástroje příkazové řádky `find`, nebo spíše zjednodušenou podmnožinu rozšířené varianty GNU find[1].

Součástí této úlohy **není** práce se souborovým systémem ani tvorba spustitelného programu. Popisy voleb proto nemusí vždy přesně uvádět jejich sémantiku – ta nás při parsování argumentů nezajímá. Přesto budeme většinu zadání psát formou dokumentace nástroje.

## Představení nástroje `find`

Nástroj `find` slouží k prohledání adresářové struktury a nalezení souborů (a adresářů) jistých vlastností. Obvyklým využitím je nalezení souborů daného jména či přípony, příliš velkých nebo příliš starých souborů apod.

Volání (našeho zjednodušeného) příkazu `find` z příkazové řádky má následující podobu:

```
find [CESTY...] [GLOBÁLNÍ VOLBY...] [VÝRAZ]
```

- `CESTY` jsou libovolný počet cest (typicky adresářů), v nichž prohledávání začíná. Není-li zadána žádná cesta, uvažuje se pracovní adresář (tj. „`.`“).
- `GLOBÁLNÍ VOLBY` jsou nepovinné přepínače měnící celkové chování nástroje.
- `VÝRAZ` je v podstatě formule výrokové logiky, která popisuje, které soubory a adresáře se považují za vyhovující a mají se objevit na výstupu.

Každou z těchto tří odrážek si přiblížíme samostatnou podsekcí.

## Argumenty

V této úloze budeme řešit jen čtení argumentů nástroje a jejich převod do haskellových datových typů.

### Oddělování argumentů

Nejprve malá odbočka k argumentům příkazové řádky obecně.

Ačkoli uživatel při spouštění programů odděluje jejich argumenty mezerami, toto **není** podoba, v níž obsah příkazové řádky vidí spouštěný program. Shell totiž programu neposkytuje jeden řetězec s celým obsahem příkazové řádky, nýbrž každý argument zvlášť.

Shell pak nabízí možnosti, jak některé mezery **ne**považovat za oddělovače argumentů. Například je možné mezeru „uvést“ zpětným lomítkem (`echo jedna\ věc`), nebo použít uvozovky (`echo 'jedna věc'`). V obou příkladech vidí příkaz `echo` jako jediný celistvý argument řetězec „`jedna věc`“. Oproti tomu volání `echo dvě věci` zpracuje shell tak, že příkaz `echo` obdrží skutečně dva argumenty: „`dvě`“ a „`věci`“.

Logicky tedy argumenty programu vnímáme jako seznam řetězců (které mohou obsahovat mezery). V této úloze budeme však počítat s poněkud nízkoúrovňovější representací a všechny argumenty budeme číst z jediného řetězce. Odděleny však nebudou mezerou, nýbrž **nulovým znakem** (`'\0'`). Za posledním argumentem nulový znak bude také. Takový přístup je známý třeba z jazyka C.

**Příklad** Volání v shellu:

```
echo první 'druhý argument' toto\ třetí   čtvrtý\   pátý ''
```

Seznam argumentů příkazu:

```
["první", "druhý argument", "toto třetí", "čtvrtý ", "pátý", ""]
```

[1] Viz `man find` nebo on-line manuálovou stránku: https://www.man7.org/linux/man-pages/man1/find.1.html

Naše parsovaná representace (toto dostanete jako vstup):

```
"první\0druhý argument\0toto třetí\0čtvrtý \0pátý\0\0"
```

Povšimněte si, jak se shell chová k vícenásobným mezerám okolo slova „čtvrtý“ a že prázdný řetězec je platný argument (a jeho výskyt vede ke dvěma sousedním nulovým znakům).

### Volby a jejich parametry

Většina voleb začíná jedním spojovníkem a pokračuje alfanumerickým jménem volby. Např. `-depth`, `-prune`. Volba musí tvořit celý argument; `-prunes`, `--prune` či `-no-prune` jsou zcela odlišné řetězce, které se nikdy nemají zpracovat jako volba `-prune` a samy netvoří platné volby.

Mnoho voleb má vlastní parametr předávaný v argumentu bezprostředně následujícím za volbou. Nepoužívá se rovnítko, ani se parametr nelepí do stejného argumentu přímo k volbě. Správně je to tedy např. volání s `-type d`, kterému bude odpovídat řetězec `"-type\0d\0"`. (Srovnejte s argumentem `'-type d'`, který by odpovídal řetězci `"-type d\0"` a tedy nesmyslné volbě s dvouslovným názvem.)

Obdobně je nutné si dát pozor na to, že závorky či vykřičník (které uvidíme později) musí být skutečně samostatný argument. Následuje-li po nich něco jiného než nulový znak, nemá závorka/vykřičník zvláštní funkci, ale je např. součástí názvu souboru.

### Cesty

Cesta je pro nás vlastně téměř libovolný řetězec bez nulového znaku uvnitř (protože ten cesty obsahovat nesmějí a neměli bychom ho jak odlišit od oddělovače argumentů). Cesta může být i prázdný řetězec.

Jedno zjednodušující omezení ale máme: cesta nemůže začínat spojovníkem. Všechny takové řetězce považujte za název volby nebo predikátu. Volání `find -lol` by tedy skončilo s chybou, že neexistuje predikát „lol“, nikoliv prohledáním všeho od cesty `./-lol`. Cestou nemůže být ani samotný vykřičník či kulatá závorka, (otevírací i zavírací), neboť ty jsou potřeba k zadávání výrazů.

### Globální volby

Mezi cestami a výrazem se mohou nacházet globální volby (v libovolném pořadí). Jsou jimi:

- `-depth` nebo `-d`: prochází adresářový strom v post-orderu.
- `-maxdepth ÚROVEŇ`: prochází nejhlouběji *úroveň* adresářů hluboko
- `-mindepth ÚROVEŇ`: uvažuje soubory až od *úroveň*-té úrovně zanoření.

`ÚROVEŇ` je číslo (viz níže upřesnění čísel). Je-li stejná volba zadána vícekrát, platí její poslední výskyt.

### Výraz

Soubory jsou vybírány na základě logického výrazu, který sestává z predikátů, operátorů a závorek, jak byste od logických formulí čekali. Trochu nezvyklé je, že posloupnost predikátů, mezi nimiž nejsou operátory, se bere jako konjunkce (tedy aby soubor vyhověl, musí platit každý z nich).

**Predikáty** Většina predikátů bere jeden parametr. Zde ho budeme oddělovat mezerou (jako by se zapsal v shellu), nezapomeňte ale, že se musí jednat o samostatný *argument*, tedy ve vstupním řetězci bude oddělen nulovým znakem. Význam predikátů uvádíme jen pro ilustraci; k řešení úkolu není potřeba.

- `-name VZOR`, `-path VZOR` – název nebo celá cesta odpovídá parametru (popsán níže).
- `-amin MINUTY`, `-atime DNY`, `-mmin MINUTY`, `-mtime DNY` – stáří posledního přístupu nebo modifikace. Parametr je *hranice* (viz níže).
- `-readable`, `-writable`, `-executable` – oprávnění uživatele na souboru/adresáři.
- `-size VELIKOST` – velikost souboru. Parametr je hranice a může mít nepovinnou, ničím neoddělenou jednotku: jedno z písmen `bcwkMG` (bloky, bajty, dvoubajtová slova, kibibajty, mebibajty, gibibajty). Výchozí je `b` (512bajtový blok).
- `-type TYP` – parametrem může být jedno z písmen `fdl` (soubor, adresář, symbolický odkaz) nebo jejich čárkami oddělovaná sekvence (např. `-type d,f`). Typy se nesmí v jedné sekvenci opakovat.
- `-prune` – vždy vyhoví, a pokud se vyhodnocuje nad adresářem, tak do něj nástroj nebude vstupovat.
- `-print` – vždy vyhoví a vypíše název souboru na výstup.

*Poznámka pro zvídavé:* Ve skutečném nástroji je `-print` sice jistým způsobem výchozí akcí, při parsování to ale nebudeme zohledňovat. Do výsledného výrazu totiž implicitně přibude jen tehdy, když není nikde ve vstupu zadán explicitně. Můžete si rozmyslet, jak byste toto pravidlo implementovali na úrovni parseru.

**Parametry predikátů** Některé parametry mají vlastní strukturu:

- **Vzor** je řetězec libovolných znaků (kromě nulového), který popisuje názvy či cesty. Některé znaky mají zvláštní význam:
  - `?` vyhoví libovolný znak,
  - `*` vyhoví libovolný (i prázdný) řetězec,
  - `\\`, `\?`, `\*` vyhoví doslova zpětné lomítko, otazník či hvězdička.
  - Jiná použití zpětného lomítka jsou považována za chybu.
- **Číslo** je posloupnost číslic a může obsahovat úvodní nuly. Je tedy vždy celé nezáporné.
- **Hranice** je číslo, které je nepovinně předcházeno ničím neoddělený znakem plus nebo spojovník. Potom například
  - `3` znamená „přesně 3“,
  - `+3` znamená „více než 3“,
  - `-3` znamená „méně než 3“.

**Operátory a závorky** Predikáty je možno pomocí operátorů seskupovat do složitějších logických výrazů. Uvedeny jsou od nejvyšší priority k nejnižší; všechny aliasy téhož operátoru mají stejnou prioritu.

- `( VÝRAZ )` – závorky kolem podvýrazu. Každá ze závorek musí být samostatný argument.
- `! VÝRAZ`, `-not VÝRAZ` – negace
- `VÝRAZ VÝRAZ`, `VÝRAZ -a VÝRAZ`, `VÝRAZ -and VÝRAZ` – konjunkce.
- `VÝRAZ -o VÝRAZ`, `VÝRAZ -or VÝRAZ` – disjunkce

Žádný z podvýrazů nemůže být prázdný.

Při testování (a zkoušení skutečného nástroje `find`) si dejte pozor na skutečnost, že závorky a vykřičník mívají v shellu zvláštní význam. Je proto nutné je ochránit zpětným lomítkem nebo zabalit do uvozovek.

### Příklady volání

*Připomenutí:* ač je zde použita „shellová notace“, na vstupu parseru `optionsParser` už uvozovky ani zpětná lomítka nebudou a mezi argumenty bude místo mezery nulový znak.

```
# Konkrétní jméno souboru (jakkoli hluboko v pracovním adresáři)
find -name hw1-sol.hs

# Všechny přímé podadresáře kořenového adresáře
find / -maxdepth 1 -type d

# Hledání videosouborů ve dvou adresářích
find media/movies media/shows -type f -a \( -name "*.mkv" -o -name "*.mp4" \)

# Symbolické odkazy nezačínající tečkou v adresářích nezačínajících tečkou
# (Explicitní -print zabraňuje výpisu souborů, pro které uspělo už -prune)
find -name ".*" -prune -or -type l -print
```

## Cílová representace

V kostře řešení naleznete definice datových typů, jimiž budete načtené argumenty representovat. Tyto typy neměňte. Pojmenování konstruktorů by mělo být dostatečně popisné; případné nejasnosti mohou osvětlit jednotkové testy, které jsou také součástí kostry.

Výsledný parser se má jmenovat `optionsParser :: Parser Options`.

## Další požadavky na parser

- Není-li zadána žádná cesta, místo prázdného seznamu bude výsledkem `["."]`, tj. relativní cesta k pracovnímu adresáři.

- Předpokládejte, že cesty a soubory nezačínají spojovníkem.
- Po sobě jdoucí hvězdičky ve vzorech načtěte jako jednu – nemá smysl to dělat jinak.
- Duplicity v parametru predikátu `-type` nejsou povoleny: např. `-type f,f` má řešení zamítnout s vhodnou chybovou hláškou (tip: `fail`).
- Více parametrů `-type` se „přeloží“ na disjunkci predikátů `TypePred`.
- Výsledek nemá obsahovat přímo zanořené negace.
- Výsledek nemá obsahovat jednoprvkové konjunkce a disjunkce.
- Výsledek nemá obsahovat prázdné konjunkce a disjunkce. Výjimkou je prázdná konjunkce na nejvyšší úrovni, která odpovídá prázdnému výrazu, jemuž vyhoví všechny soubory.
- Výsledek nemá obsahovat přímo zanořené konjunkce a disjunkce (např. `And [And x, Or y]` má být kanonisováno na `And (x ++ [Or y])`.

## Tipy

- Parsecové kombinátory vám nesmírně pomůžou (`choice`, `many1`, `endBy1`)
- Zadefinujte si pomocné parsery pro nulový znak, nenulový znak a řetězec ukončený nulovým znakem.
- Bez obav používejte `try`, ale jen u krátkých nerekursivních parserů, jako právě ty na názvy predikátů.
- Jak řešit prioritu operací? Mezilehlými vzájemně rekursivními parsery:
  - *výraz* je posloupnost oddělených *disjunktů*,
  - *disjunkt* je posloupnost nepovinně oddělených *konjunktů*,
  - *konjunkt* je predikát, *výraz* v závorkách, nebo negace jiného *konjunktu*.
- Asociativitu `And`/`Or` se snažte řešit rovnou a než až zpětně.
  - Pomůže vhodná funkce v `chainr1`.
  - Ta navíc zajistí, že vůbec nebudou vznikat jednoprvkové `And`y a `Or`y.
- Výchozí volby je možné dělat třeba přes `option` nebo `pure` jako poslední možnost `choice`.
- Občas se mohou hodit operátory `&` a `<&>` (chovají se jako flipnutý `$` a `<$>`).
- Na mnoha místech se hodí `$>`, např. `char 'b' $> Blocks`.
- Na podmíněný `fail` by se mohlo hodit `when` nebo `unless`.

## Odevzdávání

- Svou implementaci odevzdávejte do příslušné odevzdávárny v ISu.
- Na odevzdávání máte neomezeně mnoho pokusů, výsledky testů naleznete v poznámkových blocích, jak jste zvyklí z IB015.
- Po ukončení odevzdávání budou zveřejněna řešení všech studentů a zároveň vám budou přiděleni dva spolužáci, na jejichž implementaci následně napíšete peer review, tedy recenzi jejich kódu.
- Peer review se píší ve formě komentářů do recenzovaného zdrojového kódu a odevzdávají se na Aise. K tomuto ještě obdržíte e-mail s pokyny.

### Technické požadavky

- Odevzdávejte jediný soubor s příponou `.hs`.
- Řešení musí být přeložitelné GHC 9.2. Můžete si to vyzkoušet na Aise s modulem `ghc`.
- Všechny globálně definované funkce musí mít typovou signaturu.
- Řešení nesmí obsahovat hlavičku modulu.

### Moduly a rozšíření

- Můžete využít libovolné moduly z balíku `base`.
- Z balíku `parsec` si vystačíte s `Text.Parsec` a `Text.Parsec.String`.
- Řešení můžete okrášlit pomocí rozšíření `UnicodeSyntax` a balíku `base-unicode-symbols`.
- Povolenými rozšířeními jsou také `LambdaCase` a `TupleSections`.

## Hodnocení

Za úlohu můžete získat až čtyři body. Pro úspěšné absolvování předmětu potřebujete alespoň jeden bod (viz IS pro podrobné informace). Doporučujeme tedy řešení úlohy zbytečně neodkládat.

V tomto semináři nerozlišujeme body za krásu a body za funkčnost, proto berte projití testy pouze jako potvrzení toho, že něco neděláte úplně špatně, ne jako signál, že máte 4 body v kapse. Řešení, která budou

funkční, ale ošklivá, si plný počet nezaslouží. V žádném případě byste se neměli dopouštět prohřešků, před nimiž varujeme už v IB015.

Body vám udělí cvičící i se zpětnou vazbou až poté, co budou odevzdána i peer review.

Řešení tentokrát budou veřejná, takže se na ně po termínu odevzdání bude moci kdokoliv podívat. To proto, aby bylo jednodušší psát si vzájemně peer review, a částečně vás tím chceme motivovat k psaní hezčího kódu. Nebojte se pomoci si nástrojem HLint (vaši recenzenti to určitě udělají).

## Závěrečná ustanovení

- Neduplikujte kód! Využívejte vlastních i knihovních funkcí. Pomoci vám v tom může Hoogle.
- Pokud vám není něco jasné, zeptejte se v diskuzním fóru.
- V případě, že přebíráte kód odjinud, uveďte zdroj.
- Nezapomeňte, že **opisování je zakázáno** a bude postihováno podle disciplinárního řádu.