

**NAME**

rusty - malý a rychlý systém pro automatizaci překladu kódu jazyků C, C++ a pravděpodobně mnoha dalších

**SYNOPSIS**

**rusty** [-h] [--help] [--ast] [--info] [--about] [-viratwn] [-o *DIR*] [--output *DIR*] [-d *DIR*] [--dir *DIR*] [-c *COMPILER*] [--compiler *COMPILER*] [-o *DIR*] [*TARGETS*] [clean] [install] [uninstall]

**DESCRIPTION**

**rusty** je jednoduchý systém pro automatizaci překladu, který si původně zapůjčil syntaxi a koncepty ze zabudovaného systému pro jazyk C2. Syntaxe a schopnosti projektu se ovšem postupně rozrostly a nyní Rusty podporuje nejen jednoduchou kompilaci ale také inkrementální kompilaci, načítání zdrojových souborů z jedné složky, změnu kořenového adresáře projektu, podmíněné sestavování (podle operačního systému) linkování mezi cíly, příkazy pro instalaci a odinstalaci, spouštění příkazů před kompilací, přidávání přepínačů překladači a kontrolu na kterých souborech je který zdrojový soubor závislý a na kterých souborech je závislý celý projekt.

Rusty používá parsovací knihovnu mpc Daniela Holdena k načítání informací o jednotlivých cílech. Momentálně rozpoznává Rusty čtyři typy cílů: spustitelné soubory, statické knihovny, sdílené/dynamické knihovny a cíle objektového kódu, kdy nedojde k finálnímu sestavování do binárního souboru, ale pouze tvorbě objektového kódu. Rusty vyžaduje rustyfile (soubor rusty.txt). Rusty tento soubor hledá iterativně, takže není problém jej spouštět z pod-adresářů projektu. Filozofie projektu Rusty se je inspirována filozofií skupiny suckless.org. Rusty se snaží být co nejmenší (rusty.c jen něco málo přes tisíc řádků), co nejpoužitelnější a hojně využívat možnosti programovacího jazyka C.

**OPTIONS****-h, --help**

Zobrazí základní text pomoci

**--about**

Zobrazí text popisující Rusty a příklad použití.

**-c, --compiler *name***

Změní překladač použitý ke zpracování souborů.

**-i, --info**

Zobrazí informace o každém z cílů nalezeném v rustyfilu.

**-d, --dir *path***

Změní pracovní adresář před startem hledání rustyfilu.

**-r, --fullrebuild**

Znovu postaví všechny soubory bez ohledu na to jestli byly modifikovány. Užitečné pokud se nepodaří kompilace a řešením problému byla změna, která nezasahovala do zdrojových souborů.

**-t, --time**

Změří a vypíše CPU čas, jaký spuštění Rusty trvalo. Poznámka: CPU čas bývá výrazně kratší než by byl reálný čas.

**-n, --check**

Pouze zkontroluje rustyfile a nic nepřekládá/nesetavuje. Ostatní přepínače jsou stále zpracovávány. Nevztahuje se na 'run' tagy, které běží tak jako tak.

**-w, --wanted-only**

Zobrazí informace a abstraktní syntaxní strom pouze pro specifikované cíle, ne celý rustyfile. Neimplikuje přepínače -i a -a, je nutné je použít v konjunkci s -w aby se informace zobrazily.

**MODUS OPERANDI**

- 1 Rusty přečte parametry z příkazového řádku a začne hledat rustyfile.
- 2 Po nalezení rustyfilu dojde k jeho přečtení, vymazání komentářů a následnému parsování.
- 3 Na základě informací přečtených ze syntaxního stromu vytvoří Rusty cíle. 'run' tagy jsou okamžitě po přečtení spuštěny.

- 4 Rusty zkontroluje jestli pro všechny požadované cíle existují jejich soubory a vypíše chybu není-li tomu tak.
- 5 U každého souboru dojde ke kontrole, zda-li byl modifikován od předchozí kompilace. Má-li soubor `@depends()` atribut, jsou i zkontrolovány všechny soubory, zde specifikované. Dojde-li ke změně globálního závislostního souboru, je modifikovanost ignorována a všechny soubory jsou přeloženy znovu. Za tímto účelem vytváří Rusty anekdotálně minimální metadata ve složce `.rusty` v kořenovém adresáři projektu.
- 6 Po kompilaci všech souborů, pokud je úspěšná, dochází k linkovací části programu. Pro binární soubory se využije překladače bez nějakých přídatných přepínačů. Pro sdílené knihovny se používá přepínač `-shared`, očekává se, že kompilaci dodá uživatel přepínač `-fPIC` sám, uzná-li to za vhodné. Statické knihovny jsou vytvořené pomocí utility `ar`.
- 7 Pokud byl na začátku specifikován cíl `'install'` nebo `'uninstall'`, provede Rusty patřičnou operaci. Rusty povoluje jak instalaci tak odinstalaci najednou, i když se zeptá uživatele, zda-li je to úmyslné.

## LIMITACE

Rusty občas nedokáže vždy určit, zda-li se překlad povedl. Problémy také dělá, když se v `rustyfilech` vyskytuje nekonsistentní ukončení řádků. Při (od)instalaci cílů neví, jestli má dostatečná práva.

## MAPA KÓDU

- arg.h** Modifikovaný header pro zpracování přepínačů z příkazové řádky. Původně napsáno Christopherem Lohmannem. Změny zahrnují možnost dát přepínače kamkoliv do příkazu, přepínače ve formě celých slov (např. `--version`), makra pro jednoduché přepínače a integrace s Rusty. Verze `'arg.h'` obsažená v Rusty je nepoužitelná mimo něj.
- dirent.h** Implementace rozhraní `'dirent'` pro platformu Win32 napsaná Toni Ronkkem. Nutná, protože Rusty využívá rozhraní `dirent` ke čtení složek.
- mpc.c/h** Parsovací knihovna `mpc` od Daniela Holdena. Modifikovaná verze je identická s tou vloženou do projektu `c2tc`
- rusty.c** Hlavní soubor, který obsahuje drtivou většinu kódu, který není součástí vložených knihoven `mpc`, `dirent.h` a `arg.h`.
- util.h** Soubor, který definuje aliasy pro číselné typy a implementaci funkce `strdup()` pro systémy, na kterých není přítomna.

## PŘÍKLADY

Samotný Rusty má vlastní `rustyfile` v kořenovém adresáři, kterým jej jde postavit a instalovat. Standardní postup při instalaci je použití skriptu `'compile.sh'`, zkompileování Rusty pomocí této verze sebe sama, a následná instalace pomocí nové verze. To znamená zhruba tento postup:

```
$ ./compile.sh
#rustyfile projektu obsahuje dva cíle: rusty a rusty_debug. Bez specifikace by se postavily všechny cíle
najednou.
$ ./rusty rusty
$ sudo rusty install #v tomto případě zkopíruje binární soubor do /usr/bin a nainstaluje manuálovou stránku
```

Další příklad(y) lze najít ve složce `test`. Soubor `rusty.txt.concept` obsahuje i budoucí plánované funkce, které ještě nejsou implementovány. Pro pohodlí prohlížení `rustyfilů` je v kořenovém adresáři soubor `rusty.nanorc`, který obsahuje zvýrazňování syntaxe pro `rustyfil`y.

## SYNTAXE

Přesnou gramatiku `rustyfilů` lze najít v souboru `rusty.c` ve funkci `parse()`, ale tato sekce se pokusí alespoň částečně popsat tvorbu `rustyfilu`. `Rustyfile` se skládá z definice překladače a jednoho nebo více cílů. Definice překladače se vytvoří klíčovým slovem `'compiler'`, dvojtečky, jména překladače ve formě

textového řetězce a středníku. Cíl je deklarován pomocí klíčového slova 'target', identifikátoru pro cíl a dvojtečky. Identifikátor vůbec nesousíví s finálním názvem binárního souboru. Příklad:

```
compiler: "gcc";
```

**target jablko:**

**target malina:**

**target brambora:**

Tyto cíle ovšem nejsou validní, protože Rusty vyžaduje alespoň tag "name" a k tomu jeden funkční tag. Tag name specifikuje název binárního souboru, který bude vytvořen. Z hlediska formy patří mezi řetězcové tagy (tzn. vyžaduje jeden a pouze jeden textový řetězec). Každý řetězcový tag je musí být ukončen středníkem.

```
compiler: "gcc";
```

**target jablko:**

```
name: "meloun";
```

**target malina:**

```
name: "jahoda";
```

**target brambora:**

```
name: "lilek";
```

V minulém příkladu jsou všechny tagy odsazen, ale to je jen otázka stylu/čitelnosti. Rusty kompletně ignoruje bílé znaky mezi tokeny. Speciální tag je tag 'type'. Může být použit k určení typu cíle. Není vyžadován, Rusty předpokládá spustitelný soubor. Tento předpoklad ovšem závisí na tom, že funkce `calloc()` bude vynulovávat přidělenou paměť, proto je lepší vždy typ specifikovat ručně.

```
compiler: "gcc";
```

**target jablko:**

```
name: "meloun";
```

```
type: executable;
```

**target malina:**

```
name: "jahoda";
```

```
type: object;
```

**target brambora:**

```
name: "lilek";
```

```
type: libstatic;
```

Dalším tagem je tag 'flags', který přidává přepínače k překladači jak během kompilace, tak i sestavování cíle. Tag `flags` zneužívá toho, že většina překladačů jazyka C a C++ ignoruje značky, které se nehodí pro danou fázi sestavování. Tag `flags` je seznamový tag, což znamená že místo jednoho řetězce jich bere celý seznam uzavřený ve složených závorkách. Seznam musí obsahovat jeden a více řetězců, prázdné seznamy nejsou povoleny. Stejně jako ostatní tagy jsou uzavřeny středníkem

```
compiler: "gcc";
```

**target jablko:**

```
name: "meloun";
```

```
type: executable;
flags: { "-Wall", "-std=c89" };
```

**target malina:**

```
name: "jahoda";
type: object;
flags: { "-Wall", "-std=c89" };
```

**target brambora:**

```
name: "lilek";
type: libstatic;
flags: { "-Wall", "-std=c89" };
```

Tag `output` je posledním nefunkčním tagem. Mění do jaké složky se uloží výsledné binární soubory. Výchozí chování je složka `output/nazev_projektu` v kořenovém adresáři projektu. Nutno podotknout, že Rusty nevytváří tyto složky automaticky, je nutno je vytvořit předem, například pomocí `run` tagu, který je popsán níže.

```
compiler: "gcc";
```

**target jablko:**

```
name: "meloun";
type: executable;
flags: { "-Wall", "-std=c89" };
output: ".";
```

**target malina:**

```
name: "jahoda";
type: object;
flags: { "-Wall", "-std=c89" };
output: ".";
```

**target brambora:**

```
name: "lilek";
type: libstatic;
flags: { "-Wall", "-std=c89" };
output: ".";
```

řetězcové funkční tagy jsou: `'sourcedir'`, `'file'`, `'dir'`, `'depends'` a `'link'`.

Tag `sourcedir` určuje v jaké složce začíná Rusty hledat soubory specifikované pomocí tagu `file`. `Sourcedir` nemění pracovní adresář, nýbrž přidává specifikovanou cestu jako předponu k souborům.

Tag `file` může mít u sebe atribut `@depends()`, který určuje na jakých jiných souborech daný zdrojový soubor závisí ve formě odčárkovaného seznamu řetězců.

`Depends` určuje soubor, na kterém závisí celý cíl a který při změně způsobí překlad a sestavení celého cíle. Tag `depends` se zejména hodí, když má projekt headery, které jsou vloženy do většiny zdrojových souborů.

Tag `dir` najde všechny zdrojové soubory v dané složce podle přípon souborů. Seznam souborových přípon, které Rusty rozpoznává je vypsán ve funkci `read_dir()`.

Tag `link` přidá do sestavování i objektový kód jiného cíle. Pokud cíl, jehož objektový kód je přidáván, není ještě sestavený, `rusty` jej nejdříve sestaví.

```
compiler: "gcc";
```

**target jablko:**

```
name: "meloun";
type: executable;
flags: { "-Wall", "-std=c89" };
output: ".";
sourcedir: "src";
depends: "soubor.h";
file: "potato.c";
file: "potato2.c" @depends("potato.h", "util.h");
dir: "brambory";
link: mpc;
```

**target malina:**

```
name: "jahoda";
type: object;
flags: { "-Wall", "-std=c89" };
output: ".";
sourcedir: "src";
depends: "soubor.h";
file: "potato.c";
file: "potato2.c" @depends("potato.h", "util.h");
dir: "brambory";
link: mpc;
```

**target brambora:**

```
name: "lilek";
type: libstatic;
flags: { "-Wall", "-std=c89" };
output: ".";
sourcedir: "src";
depends: "soubor.h";
file: "potato.c";
file: "potato2.c" @depends("potato.h", "util.h");
dir: "brambory";
link: mpc;
```

Seznamové tagy jsou 'run', 'install' a 'uninstall'. Tag run spouští příkazy v okamžiku kdy jsou přečteny, bez ohledu na to, jestli je cíl vybrán pro kompilaci. Pokud se dostane Rusty mezi argumenty i slova install a uninstall, spustí příslušné příkazy.

```
compiler: "gcc";
```

**target jablko:**

```
name: "meloun";
type: executable;
flags: { "-Wall", "-std=c89" };
output: ".";
sourcedir: "src";
depends: "soubor.h";
file: "potato.c";
file: "potato2.c" @depends("potato.h", "util.h");
dir: "brambory";
link: mpc;
```

```

run:
{
    "echo jablko",
    "cowsay 123"
};
install:
{
    "echo instalace jablka",
    "cowsay 321"
};
uninstall:
{
    "echo odinstalace jablka",
    "cowsay 231"
};

```

**target malina:**

```

name: "jahoda";
type: object;
flags: { "-Wall", "-std=c89" };
output: ".";
sourcedir: "src";
depends: "soubor.h";
file: "potato.c";
file: "potato2.c" @depends("potato.h", "util.h");
dir: "brambory";
link: mpc;
run:
{
    "echo jablko",
    "cowsay 123"
};
install:
{
    "echo instalace jablka",
    "cowsay 321"
};
uninstall:
{
    "echo odinstalace jablka",
    "cowsay 231"
};

```

**target brambora:**

```

name: "lilek";
type: libstatic;
flags: { "-Wall", "-std=c89" };
output: ".";
sourcedir: "src";
depends: "soubor.h";
file: "potato.c";
file: "potato2.c" @depends("potato.h", "util.h");
dir: "brambory";

```

```
link: mpc;
run:
{
    "echo jablko",
    "cowsay 123"
};
install:
{
    "echo instalace jablka",
    "cowsay 321"
};
uninstall:
{
    "echo odinstalace jablka",
    "cowsay 231"
};
```

Poznámka: Na pořadí funkčních tagů nezáleží, ale pořadí hlavičkových tagů (name, type, flags, output) ano. V tomto návodu jsou specifikovány v patřičném pořadí.

## LICENCE

Rusty je licencován licencí 'Fair License'. Jedná se o absolutně minimální licenci, což jde v duchu s filozofií projektu Rusty (co nejvíce funkčnosti s co nejméně řádky kódu). Plné znění je v souboru LICENSE a zde:

Fair License

Lukáš Hozda(c), 2015-2017

Usage of the works is permitted provided that this instrument is retained with the works, so that any entity that uses the works is notified of this instrument.

DISCLAIMER: THE WORKS ARE WITHOUT WARRANTY.