GNU Tool Chain

- = balík GNU nástrojů pro vývoj aplikací napříč platformami a OS:
  - Unix: Unix, Linux, Solaris, FreeBSD, Mac OS-X
  - PC: Win32
  - Playstation3
  - ARM (WinARM – http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects/#winarm)
  - AVR (WinAVR – http://winavr.sourceforge.net/)
  - ...
- sestává z:
  - GNU make
  - GNU Compiler Collection (GCC)
  - GNU Binutils: linker, assembler a další nástroje
  - GNU Debugger (GDB)
  - GNU C Library
  - ...
- MinGW (http://www.mingw.org/
  - Minimalist GNU for Windows
  - = port GCC pro Win32 (včetně free Win32 API)

GCC

- http://gcc.gnu.org/
- GNU Compiler Collection – soubor překladačů pro:
  - C, C++(g++.exe), Objective-C, Objective-C++, Java (gcj.exe), Fortran (g77.exe), Ada
- snadná přenositelnost na jiné systémy, vysoce kvalitní
- možnost cross compilace – překladač běží na jiné než cílové platformě
- cílové architektury:
  Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64 ,IA-64, MorphoSys, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, System/390/zSeries, SuperH, SPARC, VAX, ...

Příkazová řádka (MinGW)

```
gcc.exe [parametry] soubor [soubor, ...]
```

- help na příkazové řádce: `gcc.exe --help`
- lze zastavit v kterékoli části zpracování ZK:
  - po preprocesoru `-E`
  - po kompilátoru `-c`
    - *.c (*.S) překládá na *.o

        - Příklad

```
gcc.exe -c funkce.c
```

    - preprocesor + kompilátor (assembler) + linker – bez `-c`, `-E`
- specifikace jména výstupního souboru:
    - `-o soubor`
    - výjimka – ???.c vždy automaticky na ???.o
    - pro linker nutno vždy, jinak a.exe

```
gcc.exe soubor.c -o soubor.exe
```

- určení standardu jazyka C
    - ANSI C `-ansi` (`-std=c89`)
    - GNU 89 `-std=gnu89`
        - default
        - = ANSI + něco z C99
    - C99 `-std=c99`
        - nepodporuje zcela
- míra chybových hlášení
    - pouze syntaktické chyby `-fsyntax-only`
    - chyby + varování dle ISO C `-pedantic`
    - + časté programátorovi omyly `-Wall`
- optimalizace kódu
    - vypnuto (default) `-O` (`-O0`)
    - optimalizace `-O1`
        - pouze optimalizace příliš nezvětšující dobu kompilace
    - optimalizace+ `-O2`
    - optimalizace++ `-O2`
    - optimalizace na velikost `-Os`
        - = část `-O2` + něco navíc
- přidání informací pro debugger (nutno vypnout optimalizace)
    - ve formátu COFF `-g`
    - pro GDB `-gdb`
- kontrola preprocesoru
    - `#define MAKRO` `-d MAKRO`
    - `#define MAKRO hodnota` `-d MAKRO=hodnota`

Časté omyly

```
gcc.exe soubor.c -o soubor.exe
```

- kromě soubor.c zkompiluje i všechny *.c, zmíněné ve vložených *.h souborech v soubor.c – nikoli

```
gcc.exe s1.c s2.c s3.c
```

- kompilátor při překladu s?.c prohlíží obsah ostatních s?.c – nikoli, každý s?.c překládán samostatně

GNU Make

- http://www.gnu.org/software/make/manual/html_node/index.html
- původně make – AT&T 1977
  - pomocný nástroj pro UNIX
- zjednodušuje a automazuje překlad velkých projektů – make spouští překladač, linker
- řízen textovým (ASCII) souborem – „makefile“
  - popisuje závislosti mezi soubory
  - OS udržuje čas poslední modifikace souboru – lze překládat jen nejnutnější
- konkrétní implementace make – vždy nutno prostudovat syntaxi
- Microsoft VS – nmake.exe
- struktura makefile:
  - komentáře
  - pravidla
  - cíle
  - proměnné
  - příkazy

Komentáře

- pouze řádkové

```
# toto je komentář
# toto je na další řádce komentář
```

- složitý makefile = bohaté komentáře – dokumentační funkce makefile

Pravidla (Rules)

- = postup vedoucí ke splnění daného cíle

```
cíl: seznam závislostí
<TAB> příkaz
```

  - cíl: soubor
- příklady

```
prog.exe: prog.o funkce.o
	gcc.exe prog.o funkce.o -o prog

funkce.o: funkce.c \
	funkce1.h
	gcc.exe -c funkce.c
```

Cíle (Targets)

- = co je výsledkem make

- počet neomezen – různá jména
- typy:
  - fyzický soubor – viz výše
  - symbolické cíle – pouze název akce

Symbolické cíle

- důvody:
  - vytvoření několika souborů
  - kombinace více akcí
  - cílem je činnost (ne soubor)
- žádné příkazy na další řádce
- vytvoření několika souborů

```
all: soubor1.exe soubor2.exe
soubor1.exe:
	# akce
soubor2.exe:
	# akce
```

- cílem je činnost (spouštění příkazů OS)

```
clean:
	rm program.exe rm *.o
```

- kombinace více akcí (cílů)

```
# Default target.
all: begin gccversion $(TARGET).elf $(TARGET).hex $(TARGET).eep \
	$(TARGET).lss $(TARGET).sym sizeafter finished end
```

Pořadí cílů

- na pořadí záleží
- příklad: vytvořit .exe z .c jako defaultní cíl make

```
prog.exe: prog.o
	gcc.exe prog.o -o prog
prog.o: prog.c
	gcc.exe -c prog.c
```

```
prog.o: prog.c
	gcc.exe -c prog.c
prog.exe: prog.o
	gcc.exe prog.o -o prog
```

Proměnné (Variables)

- cca `#define`
- definice

```
jméno = hodnota
```

- jméno – všechny znaky kromě : `#`, `=`, `<MEZERA>`
- použití

```
$(jméno)
```

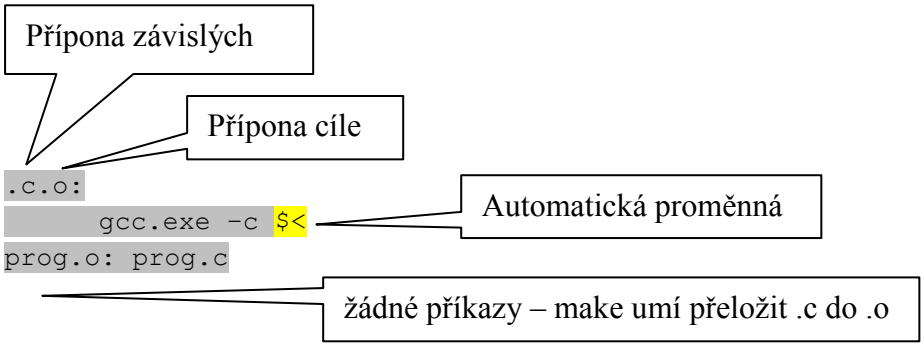
- příklad

```
objects = program.o foo.o utils.o
program: $(objects)
	cc -o program $(objects)
$(objects): defs.h
```

- `$` v textu – nutno `$$`

Implicitní pravidla (Implicit Rules)

- pravidla pro vznik souborů s určitými příponami



Automatické proměnné (Automatic Variables)

- (hlavně) ve spojení s implicitními pravidly
- dosazeny by make při každém spuštění pravidla
- výběr:

`$@`

  - jméno cíle pravidla (včetně cesty + přípony)

`$<`

  - jméno prvního závislého (včetně cesty + přípony)

`$^`

  - jména všech závislých (včetně cesty + přípony) oddělených mezerami

`$?`

  - jména všech závislých (včetně cesty + přípony) oddělených mezerami, která jsou novější než cíl
  - příklad – do knihovny archivuje jen novější

```
lib: foo.o bar.o lose.o win.o
	ar r lib $?
```

Předdefinované proměnné

- (někdy) k dispozici by make (nespoléhat na to!!!)
- pokud make nedává – doporučeno vytvářet a používat (čitelnost makefile)

`CC`

  - jméno kompilátoru C včetně cesty

`AS`

  - jméno assembleru včetně cesty

`MAKE`

  - Příkazová řádka, se kterou bylo make spuštěno

`MAKEFLAGS`

  - parametry příkazové řádky, se kterou bylo make spuštěno

`DEBUG`

  - jméno kompilátoru C včetně cesty

`CFLAGS`

  - parametry C kompilátoru

`LFLAGS`

  - parametry linkeru

- Příklad:

```
OBJS = MovieList.o Movie.o NameList.o Name.o Iterator.o
CC = g++
DEBUG = -g
CFLAGS = -Wall -c $(DEBUG)
LFLAGS = -Wall $(DEBUG)

p1 : $(OBJS)
	$(CC) $(LFLAGS) $(OBJS) -o p1
```

Speciální cíle

- jméno spec. cíle v pravidlu = speciální význam
- seznam všech viz. viz. http://www.gnu.org/software/make/manual/html_node/Special-Targets.html#Special-Targets

„Phony“ cíle

- pro symbolické cíle
- důvody
  - pro urychlení práce (nezkoumají se implicitní pravidla)
  - pokud v projektu jméno cíle (soubor) = jméno symb. cíle – zamezení konfliktu jmen (clean „soubor“ vs. clean „mazání“)
- příklad a použití:
  - místo:

```
clean:
	rm program.exe rm *.o
```

- lepší:

```
.PHONY: clean
clean:
	rm program.exe rm *.o
```

- více phony cílů

```
.PHONY: cleanall cleanobj cleandiff
cleanall : cleanobj cleandiff
	rm program
cleanobj:
	rm *.o
cleandiff :
	rm *.diff
```

Příkazy OS v makefile

- Win32 = příkazy konzole Win32 (MS-DOSu)
- Linux, Unix – shell
- Po provedení se vrací řízení zpět k make
- použití: archivace, obsluha souborového systému (mazání, kopírování, ...), spuštění utilit (MCU technika – programátory)
- příklady:

```
echo Spoustim mazani          # konzole: echo Spoustim mazani
@echo Spoustim mazani         # konzole: Spoustim mazani
mkdir ahoj                    # konzole: mkdir ahoj
@ren pokus.exe C:\nic.exe     # konzole:
```

Spuštění make

- bez parametrů – hledá soubor „makefile“ nebo „Makefile“, provede první cíl
- výpis všech implicitních pravidel, automatických a předdefinovaných proměnných

```
c:\make -p
```

v adresáři nesmí být žádný makefile

- s parametry – jméno cíle

```
c:\make clean
```

Spolupráce programu a OS

- uživatelský program může:
  - převzít od OS parametry při spuštění
  - vracet OS informace po skončení
    - return 0 → konec programu OK
    - return >0 → program skončil chybou
  - spouštět jeho příkazy
- vhodné pro dávkovém zpracování (*.bat soubory)

Spouštění příkazů OS

```
int system (const char *string);
```

- Spustí příkazový interpret a předá mu řetězec `string`.
- např. `system("PAUSE"); system("format c:"); system("mujprogram.exe")`

Parametry příkazové řádky

- funkce `main()` může mít parametry – naplní OS hodnotami z příkazové řádky:

  ```
  int main(int argc, char* argv[])
  ```

  - `argc` = počet parametrů (včetně jména programu)
  - `argv` pole řetězců s parametry (včetně jména programu)
    - oddělené mezerami
    - „“ = jeden řetězec
- příklad: `program -124.33 "ahoj mami" 10`

  ```
  argc == 4;
  argv[0] == "program"
  argv[1] == "-124.33"
  argv[2] == "ahoj mami"
  argv[3] == "10"
  ```

- Příklad – vypíše parametry příkazového řádku

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

  int main(int argc, char* argv[])
  {
        int i;
        for (i=1; i<argc; i++)
              printf("%d. parametr: %s\n", i, argv[i]);
        system("PAUSE");
        return 0;
  }
  ```

Funkce s proměnným počtem parametrů

- = počet parametrů není pevně dán (omezen)
- korektní práce → jeden parametr = počet proměnných
- např. `printf(const char *format, ...)`
  - ve `format` počet `%` = počet dalších parametrů
- definice funkce s PPP
  - `typ funkce(typ p1, typ p2, ...)`
  - `...` = výpustka
  - min. jeden arg musí být pevně dán (počet dalších param)

práce s PPP

- v stdarg.h datový typ `va_list` → pointer na pole parametrů
- a několik maker:

  `void va_start(va_list ap, last);`

  - inicializuje `ap` na první PP, `last` je identifikátor posledního známého parametru

  `type va_arg(va_list ap, type);`

  - vrací hodnotu akt. PP typu `type` ze seznamu `ap` a inkrementuje jej

  `void va_end(va_list ap);`

  - resetuje `ap`
- princip na příkladu:

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

void funkce(int pocet, ...)
{
	int i, aktParam;
	va_list ap;

	va_start(ap, pocet);		// posledni znamy parametr
	for (i=0; i<pocet; i++) {
		aktParam = va_arg(ap, int);
		// zpracovani dalsich parametru, napr.
		printf("%d\n", aktParam);
	}
	va_end(ap);			// nutne ukonceni fce s PPP
}

int main(void)
{
	funkce(5, 1, 2, 3, 4, 5);
	funkce(2, 10, 20);
	return 0;
}
```