Laboration 2 - `mmake`

Din uppgift är att implementera en minimal version av Unix-verktyget `make` i C. `make` är ett verktyg som används för att automatisera byggandet av program. `make` läser in en *make*-fil som innehåller regler som avgör när olika delar i programmet behöver byggas om.

Ett exempel på en *make*-fil är följande som kan användas för att bygga programmet `mexec` från obligatorisk uppgift 1:

```
mexec: mexec.o mexec_parser.o
        gcc -o mexec mexec.o mexec_parser.o

mexec.o: mexec.c mexec_parser.h
        gcc -c mexec.c

mexec_parser.o: mexec_parser.c mexec_parser.h
        gcc -c mexec_parser.c
```

Denna *make*-fil används för att bygga en exekverbar fil vid namn `mexec`. För att kunna bygga `mexec` behöver den ha tillgång till objektfilerna *mexec.o* och *mexec_parser.o*. Objektfilerna *mexec.o* och *mexec_parser.o* har i sin tur varsin regel för att kompileras utifrån motsvarande C-filer.

Uppgiften ska utföras enskilt.

Gränsyta

Programmet har följande synopsis:

```
mmake [-f MAKEFILE] [-B] [-s] [TARGET ...]
```

Programmet ska stödja tre stycken frivilliga flaggor. Flaggan `-f` används för att använda en annan *make*-fil än filen som heter *mmakefile* i nuvarande katalog (som programmet exekverar i). Flaggan `-B` används för att tvinga ombyggnation trots att filer inte har uppdaterats. Flaggan `-s` används för att inte skriva ut kommandon till *standard output* när de exekveras.

Programmet kan också ta in en lista av mål (*TARGET*) som vanliga argument. Dessa argument avgör vilka mål som ska byggas. Om inget mål specificeras kommer programmet bygga standardmålet (*default target*).

Filformat

Make-filen som programmet ska läsa in består av en eller flera regler som har följande format:

```
target : [PREREQUISITE ...]
        program [ARGUMENT ...]
```

En regel består av två rader. Första raden får inte börja med blanktecken och innehåller målet vilket är namnet på det som regeln ska uppfylla. Därefter kommer ett kolon och en lista av beroenden (*prerequisite*), vilket är namn på saker som måste vara uppfyllda innan regeln kan utföras (i denna uppgift filer som målet beror på). Beroenden i listan separeras av blanktecken. Listan av beroenden avslutas med ett nyradstecken.

Nästa rad börjar med ett tab-tecken följt av ett kommando som ska utföras så snart som beroendena är uppfyllda. Ett kommando är uppbyggt av namnet på det kommando (i denna uppgift ett program) som ska exekveras följt av en lista med argument till kommandot. Argumenten i listan separeras av blanktecken. Listan av argument avslutas med ett nyradstecken.

Blanktecken (förutom nyradstecken) har ingen betydelse så länge det inte är första tecknet i en rad eller används för att separera element i en beroendelista eller en argumentlista.

Målet för regeln högst upp i filen kallas standardmål (*default target*) och är det som kommer byggas ifall användaren av programmet inte specificerat vad som ska byggas.

Beteende

När `mmake` startas så ska det ladda in alla regler från *make*-filen *mmakefile* i nuvarande katalog eller filen som är specificerad med `-f` flaggan. Om användaren anger ett mål som inte existerar ska ett lämpligt felmeddelande skrivas ut.

`mmake` ska fungera som en begränsad version av Unix-kommandot `make`. Programmet ska använda filernas tidstämpel för senaste modifiering för att avgöra vilka delar av programmet som behöver byggas om. Programmet ska göra detta genom att titta på reglerna i *make*-filen. Ett mål ska endast byggas ifall någon av följande tre situationer gäller:

- Målet existerar inte
- Tidpunkt då ett beroende modiferades är senare än tidpunkten då målet modiferades
- Flaggan `-B` används

Ifall ett mål behöver byggas om ska kommandot i dess regel exekveras. Om inte `-s` flaggan används ska kommandot också skrivas ut på *standard output*. Om det finns en regel för något av beroendena ska `mmake` utföra samma logik rekursivt för att uppdatera beroendet *innan* programmet avgör om den nuvarande regeln behöver utföras.

Ifall något av beroendena saknas och det inte finns någon regel för att bygga det ska ett lämpligt felmeddelande skrivas ut på *standard error*. Ifall ett kommando som exekveras misslyckas (inte returnerar `EXIT_SUCCESS`) ska *mmake* avslutas med `EXIT_FAILURE`.

Hjälpmaterial till uppgiften

Till din hjälp tillhandahålls filerna *parser.c* och *parser.h* som implementerar en parser av filformatet som ska användas i lösningen. Filerna återfinns tillsammans med specifikationen på samma sida i Canvas. Gränsytan till parsern består av följande funktioner (som finns dokumenterade i *parser.h*):

```
makefile *parse_makefile(FILE *fp);

const char *makefile_default_target(makefile *make);

rule *makefile_rule(makefile *make, const char *target);

const char **rule_prereq(rule *rule);

char **rule_cmd(rule *rule);

void makefile_del(makefile *make);
```

Funktionen `parse_makefile` används för att parsa en öppnad *make*-fil till en datastruktur av typen `makefile`. Denna funktion returnerar `NULL` om filen inte kunde parsas.

Funktionen `makefile_default_target` returnerar vad standardmålet är för en *make*-fil.

Funktionen `makefile_rule` returnerar en datastruktur av typen `rule` med regeln för att bygga ett mål i *make*-filen eller `NULL` om en sådan regel inte existerar.

Funktionen `rule_prereq` returnerar en `NULL`-terminerad array med filnamn som utgör beroendena för en regel.

Funktionen `rule_cmd` returnerar en `NULL`-terminerad array med namnet på det aktuella kommandot, samt argumenten till kommandot, som ska exekveras för att utföra en regel (tänk på att första argumentet är namnet på det aktuella kommandot).

Funktionen `makefile_del` används för att frigöra minnet från en datastruktur av typen `makefile`.

Begränsningar

Programmet behöver inte ha större noggrannhet än en sekund när det kommer till att avgöra om en fil behöver byggas om. Programmet behöver inte heller hantera cirkulära beroenden i en *make*-fil. Eftersom det är en begränsad version av `make` ska programmet inte hantera saker som variabelexpansion eller *wildcards*.

Kompilering

Programmet ska gå att kompilera utan varningar med hjälp av `make` och den *make*-fil du levererar tillsammans med programmet. Separatkompilering, som även tar hänsyn till alla eventuella *header*-filer, ska användas. Observera att din *make*-fil ska skrivas manuellt och ej genereras av något verktyg.

Vid kompilerig ska åtminstone följande flaggor användas för alla steg:

```
-g -std=gnu11 -Werror -Wall -Wextra -Wpedantic
-Wmissing-declarations -Wmissing-prototypes -Wold-style-definition
```

Kontroll av anrop

Alla funktionsanrop som returnerar eller på annat sätt ger tillbaka en indikation för hur anropet gick ska kontrolleras. Detta innefattar till exempel `malloc` som returnerar `NULL` om minne ej gick att allokera.

Dessa funktioner behöver du *inte* kontrollera:

- `write`, `printf` och `fprintf` (eller annan utskriftfunktion)
- `close` och `fclose`
- `free`

Om ett anrop misslyckats ska programmet skriva ut ett lämpligt felmeddelande på *standard error* för att sedan avsluta med returkoden `EXIT_FAILURE`. Om anropet ändrar värdet på `errno` vid fel ska `perror` användas för att skriva ut felmeddelandet.

Rapport

Förutom kod och en *Makefile* ska en rapport lämnas in. Den rapport som ska skrivas ska ha formen av en vanlig rapport och innehållsmässigt vara en "utökad" *man*-sida. Målgruppen för rapporten är studenter med motsvarande kunskaper i programmeringsspråket C men som inte läst kursen. Rapporten ska innehålla följande delar:

- Försättsblad
- Namn (*Name*)
- Synopsis (*Synopsis*)
- Beskrivning (*Description*)
- Lösning
- Exit-status (*Exit status*)
- Diskussion och reflektion

Försättsbladet ska vara en separat sida och innehålla titel, ditt namn, din cs-användare samt kursens namn.

För att se vad delarna namn, synopsis, beskrivning och exit-status bör innehålla får du studera ett antal olika *man*-sidor.

Lösningsdelen är det som är den huvudsakliga utökningen jämfört med en *man*-sida. I denna del ska du ge en övergripande beskrivning av din lösning, inkluderande en beskrivning av huvudalgoritmen för programmet (tänk på vad uppgiften syftar till att behandla så du inte abstraherar bort viktig funktionalitet från din huvudalgoritm).

I diskussions- och reflektionsdelen kan du skriva om problem som du stötte på när du gjorde uppgiften, vad du tyckte om den och övriga åsikter som du vill framföra.

Övriga krav på rapporten:

- Väl strukturerad
- Väl formaterad
- Väl formulerad
- Utan språkliga fel
- Eventuella tabeller, figurer, etc ska hanteras korrekt
- Eventuella referenser ska hanteras korrekt enligt Harvard-stilen (https://www.umu.se/bibliotek/soka-skriva-studera/skriva-referenser/)

Övriga krav på lösningen

Förutom de krav som nämns i ovanstående beskrivning ska även programmet uppfylla följande krav:

- Din lösning ska var så enkel som möjligt. Det gäller både algoritmiskt och kodmässigt. Din lösning ska enkelt förstås av annan student som läser denna kurs och som tagit till sig av kursmaterialet som examineras i denna obligatoriska uppgift.
- Programmet ska använda sig av funktionerna:
  - `stat` eller `lstat`
  - `fork`
  - `execvp` (eller annan variant av `exec`)
  - `wait` (eller annan variant av `wait`)
  - `getopt` (inklusive `optind`) för att läsa in argument.
- Programmet får inte ha några minnesläckor, använd verktyget *valgrind*. Detta gäller även vid felhantering.
- Programmet får inte använda sig av globala variabler.
- Ändringar får inte göras i de tillhandahållna filerna.
- All eventuell utskrift i samband med felhantering ska ske till *standard error*.
- Koden ska ha god abstraktion, ha bra struktur, ha bra modularisering, samt ha en lämplig uppdelning i funktioner.
- Koden ska skrivas med god kodkvalité (indentering, variabelnamn, kommentarer, funktionskommentarer, etc.).
- Programmets huvudfil ska heta *mmake.c*.
- Inlämningen ska klara alla tester i labres.

Bedömningsmall

Tillsammans med denna specifikation finns den bedömningsmall som kommer att användas vid bedömningen av din inlämning (ligger i Canvas på samma sida som specifikationen).

Inlämning

Kod, *make*-fil och rapport lämnas in via labres innan utsatt deadline.