Lab 5: Biblioteci in C/Makefile

Formatori:
Tutor: Stiegelbauer Paul
Tutor: Iovanovici Alexandru
+1

Data de începere a cursului:
19.02.2024
Utilizatori înscriși
Calendar
Note

Lab 5: Biblioteci in C/Makefile

Utilitarul make si fisierele Makefile
(Material si exemple adaptate din cursul lui Alin Anton)

Sunt utilitare din universul GNU folosite pentru a automatiza procesul de *build* al aplicatiilor dezvoltate in C/C++ (exista solutii similare si pentru alte limbaje precum Java, C# etc). Scopul este de a "scripta" regulile si "comenzile" necesare pentru a obtine fisiere non-sursa (pot fi executiabile sau binare-obiect) din unul sau mai multe fisiere sursa si eventual de a "instala" programul obtinut pe anumit sistem de operare.

Utilitatul **make** extrage informatiile si regulile necesare pentru crearea binarelor din fisierul **Makefile** care trebuie creat de dezvoltator. Structura de baza a unui fisier Makefile este:

```
target: dependencies
[tab] system_command
```

Spre exemplu un fisier Makefile simplu este:

```
all:
[tab] gcc -Wall -O3 -ansi -pedantic main.c hello.o string.o -o hello.x
```

care compileaza folosind gcc cu o serie de argumente (ce face ficare?!?) si "linkeaza" cod binar (obiect) din fisierele "hello.o" si "string.o" in construirea fiserului "final", hello.x. Daca creati acest fisier si executati "comanda" **make** in acelasi folder cu **Makefile**-ul si fisierele surse si obiect referite, se va executa implicit regula **[all]**.

O structura generala si mai flexibila, de la care puteti porni in implementarea lucrarii de laborator este cea de mai jos:

```
# the compiler: gcc for C program, define as g++ for C++
CC = gcc
# compiler flags:
# -g adds debugging information to the executable file
# -Wall turns on most, but not all, compiler warnings
CFLAGS = -g -Wall -O3

# the build target executable:
TARGET = hello

# the files cleanup
RM = rm -f

all: $(TARGET)

$(TARGET):
    $(CC) $(CFLAGS) -c string.c
    $(CC) $(CFLAGS) -c main.c
    $(CC) $(CFLAGS) -o $(TARGET).x string.o main.o

clean:
    $(RM) $(TARGET).x
```

Astfel aveti un antet in care definiti niste "variabile" precum CC (calea sau numele utilitarului de C-compiler) si o lista de CFLAGS (Compiler flags). De asemenea definiti o variabila cu numele TARGET care va fi folosita mai jos. In regula **all**se mentioneaza ca trebuie executat ceea ce este in sectiunea TARGET (prin utilizarea simbolului $ si in paranteze numele blocului, iar $(TARGET) specifica precis setul de instructiuni care trebuie executat ca parte a procesului de *make-all* in speta compilarea celor doua fisiere sursa (string.c si main.c), pe fiecare cu setul de argumente din CFLAGS urmata de linkarea fisierelor obiect produse si obtinerea fisierului "executabil" cu numele hello.x

Crearea si gestionarea de biblioteci software in C

Se foloseste pentru a "impacheta" mai multe fisiere obiect intr-un fisier care spre exemplu permite reutilizarea de functionalitate (fspre exemplu iserul biblioteca pentru operatii matematica are antetul declarat in fisierul math.h).

Spre exemplu putem crea o biblioteca de utilitare simple de liceu de cu numele **hs_utils** in felul urmator:

- cream fisierul hs_utils.c cu urmatorul continut

```
/* hs_utils.c */
unsigned estePrim(unsigned long long n) {
      if (n%2==0){
         return 0;
      }
      //restul codului
}
```

-iar mai apoi cream fisierul hs_utils.h in care declaram functiile din hs_utils.c

```
/* hs_utils.h */
unsigned estePrim(unsigned long long);
```

Bibliotecile pot fi "linkate" in executabil in mod static, pot fi apelate ca si shared-objects sau pot fi incarcate in mod dinamic in timpul executiei (ca si dll-urile in mediul MS Win).

Astfel am putea folosi biblioteca creata mai sus intr-un program in felul urmator:

```
/* main.c */
#include "hs_utils.h"
void main(void) {
   if (estePrim(23)){
     //...
   }
}
```

in care solicitam includerea la preprocesare a fisierului **hs_utils.h** care permite compilatorului sa aiba acces la antetul functiei **estePrim** urmand ca implementarea ei sa fie accesibila la linkare din fisierul obiect obtinut prin compilarea lui **hs_utils.c**. Pentru automatizarea regulilor de build se poate folosi **make** precum mai sus.

Lucrare de laborator

1. Implementati biblioteca descrisa succint sub numele de **hs_utils** (fisierele **hs_utils.c** si hs_utils.h); Extindeti adaugant functiile int ePalindrom(unsigned) si unsigned getFibboTerm(unsigned n). Cele doua functii verifica daca respectiv argumentul este palindrom

si returneaza cel de-al n-ulea termen al sirului lui Fibbonaci.
2. Testati biblioteca hs_utils intr-un fisier client;
3. Creati un fisier Makefile pentru gestiunea procesului de build pentru biblioteca hs_utils si aplicatia client.
4. Implementati o biblioteca si fisierul Makefile aferent pentru structura de date Coada (conform cu cele prezentate in cursul 4 si exersate in laboratorul anterior).
5. Implementati "Aplicația 3" din "Lab3: Structura de date stiva si coada" folosind biblioteca de Stiva de la Curs.

Resurse specifice

1. Anton, A., & Creţu, V. (2016). *C programming techniques: Laboratory assignments* (Calculatoare-Informatică 100). Timişoara: Editura Orizonturi Universitare.
2. Klemens B. 21st Century C: C Tips from the New School. " O'Reilly Media, Inc."; 2014 Sep 27.
3. GeeksForGeeks
4. TutorialsPoint

◄ Predare Proiect 1

Sari la...

Test1 Lab restant ►