Podstawy programowania

Laboratorium 12.

Cel:

- Opanowanie podstaw tworzenia i wykorzystania modułów i bibliotek w C

Zajęcia:

1. Utworzenie katalogu roboczego *lab_12*
2. Skopiowanie ze strony przedmiotu do podkatalogu np. *lista_1* pliku *lista_powiazana.tgz*, wyodrębnienie plików *lista_powiazana.c* i *Makefile*
3. Zaznajomienie ze strukturą pliku *Makefile* i zasadami działania programu *make*:
   - część pierwsza *Makefile*: definicje symboli wykorzystywanych później w procesie kompilacji
     - symbole oznaczające: kompilator, konsolidator (linker), opcje kompilacji, opcje linkowania, niestandardowe położenia plików nagłówkowych, niestandardowe położenia bibliotek i opcje dołączania bibliotek, (ewentualnie także program do archiwizowania bibliotek i inne)
   - część druga *Makefile*: zestaw celów kompilacji i linkowania razem z poleceniami wykonywanymi dla każdego celu
     - dla pojedynczego celu:
       - linia 1: nazwa celu (np. nazwa pliku binarnego będącego celem linkowania, nazwa pliku pośredniego będącego celem kompilacji, nazwa celu specjalnego – *run* dla uruchomienia programu, *clean* dla wyczyszczenia katalogu (katalogów) z plików pośrednich (i ewentualnie binarnych)
       - linia 1 cd.: po nazwie celu dwukropek (:) i lista zależności – lista plików, których zmiany wymuszają realizacje poleceń związanych z celem, np. plik wykonywalny zależy od plików pośrednich (zmiana plików pośrednich wymusza linkowanie), pliki pośrednie zależą od plików źródłowych (zmiana plików źródłowych wymusza kompilację) (uwaga: *make działa rekurencyjnie, chcąc realizować cel, sprawdza jego zależności, następnie zależności zależności, zależności zależności zależności itd.)*
       - linia 2 i następne: **rozpoczynają się od znaku TAB,** każda linia zawiera polecenie do wykonania dla realizacji celu (są to najczęściej polecenia linkowania dla celów będących plikami wykonywalnymi i polecenia kompilacji dla celów będących plikami pośrednimi) – może być wiele linijek z poleceniami, aż do następnego bloku związanego z kolejnym celem
4. Uruchomienie programu, korzystając z dostarczonego pliku *Makefile*
   - obserwacja działania *make*:
     - zmiana pliku źródłowego -> konieczność rekompilacji
     - usunięcie pliku binarnego -> konieczność konsolidacji
     - wywołanie standardowe *make* (dla pierwszego celu w *Makefile*), wywołanie dla konkretnego celu *make nazwa_celu* , wywołanie dla wszystkich celów *make all*
     - modyfikacja *Makefile* (np. zmiana opcji kompilacji – poziomu optymalizacji lub tp.)
       - w momencie kiedy zmiany dotyczą tylko *Makefile*, *make* nie wykonuje żadnych działań; w celu wymuszenia rekompilacji kodu (np. z nowymi opcjami optymalizacji) konieczne jest jawne wywołanie *make clean* (usunięcie "starych plików") i ponownie uruchomienie *make*
5. Analiza struktury programu z pliku *lista_powiazana.c* – jego struktur danych i funkcji
   - celem niniejszych zajęć jest praca nad procesem tworzenia kodu, stąd istotna jest tylko formalna struktura programu, treść funkcji, to co realizują będzie badane w trakcie odrębnych zajęć
6. Utworzenie nowego katalogu np. *lista_modul*, skopiowanie do niego pliku *lista_powiazana.c*
7. W nowym katalogu, dokonanie modyfikacji programu, tak, aby wydzielić funkcje obsługujące listę powiązaną do osobnego pliku źródłowego, który można uznać za odrębny moduł programu – interakcje z takim modułem odbywają się wyłącznie za pomocą elementów zapisanych w odpowiednim pliku nagłówkowym
   - utworzenie pliku nagłówkowego *lista_powiazana.h* zawierającego elementy tworzące wyłączny interfejs korzystania z kodu funkcji realizujących implementację listy powiązanej
     - definicje typów
     - deklaracje funkcji
     - umieszczenie na początku pliku konstrukcji zabezpieczającej przed wielokrotnym

włączaniem do plików źródłowych, wykorzystującej warunkową kompilację, np.:
*#ifndef _lista_powiazana_*
*#define _lista_powiazana_*
- (nie należy zapomniec o *#endif* na końcu pliku *lista_powiazana.h*)

- przeniesienie funkcji *main* do osobnego pliku *main.c*
  - plik *main.c* poza włączeniem plików nagłówkowych standardowych bibliotek (np. *stdlib.h* i *stdio.h*) powinien także dołączać plik nagłówkowy *lista_powiazana.h* (składnia *#define* – patrz wykład)
    - plik *main.c* (po włączeniu pliku nagłówkowego *lista_powiazana.h)* zawiera teraz definicje typów związanych z listą powiązaną i deklaracje procedur obsługujących listę
- usunięcie definicji typów z pliku *lista_powiazana.c* (definicje funkcji pozostają jako główna treść *lista_powiazana.c)* – włączenie do pliku *lista_powiazana.c* pliku nagłówkowego *lista_powiazana.h*
  - plik *lista_powiazana.c* (po włączeniu pliku nagłówkowego *lista_powiazana.h)* zawiera teraz definicje typów związanych z listą powiązaną oraz deklaracje i definicje procedur obsługujących listę
- modyfikacje pliku *Makefile*
  - uzupełnienie bloku (reguły) dla pliku wykonywalnego *zabawa* o zależność od pliku *main.o* i linkowanie z plikiem *main.o* w komendzie konsolidacji
  - utworzenie nowego bloku (reguły) dla pliku *main.o* (zależnego od pliku *main.c* i pliku *lista_powiazana.h* (każda modyfikacja interfejsu wymaga rekompilacji kodu – dla sprawdzenia czy wykorzystanie modułu pozostaje w zgodzie z jego nowym interfejsem)
  - uzupełnienie bloku dla pliku *lista_powiazana.o* o zależność od *lista_powiazana.h*
- uruchomienie programu, sprawdzenie poprawności działania
  - modyfikacja kolejnych plików źródłowych (*main.c, lista_powiązania.c i lista_powiązania.h)* i każdorazowe sprawdzanie operacji wykonywanych przez *Makefile*

---------- 3.0 ----------

8. Utworzenie nowego katalogu np. *lista_biblioteka*, skopiowanie do niego zawartości katalogu *lista_modul*
9. W nowym katalogu, dokonanie modyfikacji programu, tak, aby z pliku pośredniego *lista_powiazana.o* utworzyć bibliotekę i nadać całemu programowi w *main.c* standardową strukturę kodu źródłowego korzystającego z własnych bibliotek (bibliotekę należy traktować jak moduł, z którym interakcje odbywają się wyłącznie za pomocą elementów zapisanych w odpowiednim pliku nagłówkowym)
   - zarchiwizowanie pliku *lista_powiazana.o* do postaci biblioteki statycznej *liblista_powiazana.a* za pomocą komendy
     - *ar -ru liblista_powiazana.a lista_powiazana.o*
   - utworzenie katalogu *include* – przeniesienie pliku *lista_powiazana.h* do katalogu *include*
   - utworzenie katalogu *lib* – przeniesienie pliku biblioteki *liblista_powiazana.a* do katalogu *lib*
   - utworzenie katalogu *src* – przeniesienie plików *main.c i Makefile* do katalogu *src* (w rozbudowanych programach katalog *src* może mieć złożoną budowę, składać się z wielu podkatalogów – zawierających np. pliki z kodem źródłowym dla różnych modułów programu; w takim przypadku plik *main.c* może znajdować się w podkatalogu *src/main*)
   - usunięcie wszystkich plików z katalogu *lista_biblioteka* – powinien zawierać tylko podkatalogi *src, include i lib*
   - modyfikacja pliku *Makefile* (w katalogu *src*)
     - odkomentowanie linijek definiujących symbole *INC i LIB* – przekazujących kompilatorowi informacje o położeniu plików nagłówkowych oraz plików bibliotek i nazwach bibliotek
       - uwaga na składnię dołączenia biblioteki – nazwa musi zawierać człon *lib*, który nie pojawia się w opcji kompilatora dołączenia biblioteki *-l*
     - usunięcie wszelkich odwołań i reguł dla pliku *lista_powiazana.o*
     - zmiana zależności *main.c* wskazująca na nowe położenie *lista_powiazana.h (../include/lista_powiazana.h)*
   - kompilacja, uruchomienie programu (w katalogu *src*)
     - jako zadanie dodatkowe można rozważyć utworzenie odrębnych katalogów (na poziomie *src, include i lib*):

- *obj* – do przechowywania plików kodu pośredniego
- *bin* – do przechowywania ostatecznie utworzonego kodu wykonywalnego

---------- 4.0 ----------

1. Utworzenie katalogu *zadanie_dodatkowe*
2. Dla kodu z laboratorium 11 (plik *tekst_wieloliniowy.c*) stworzenie struktury składającej się z samej funkcji *main*, w pliku *main.c* oraz biblioteki zawierającej wszystkie opracowane podczas laboratorium funkcje (tyle funkcji i udało się podczas laboratorium zaimplementować).
   - opracowane funkcje można przenieść do odrębnego katalogu *tekst_wieloliniowy* (np. dalej w pliku pod nazwą *tekst_wieloliniowy.c*) i utworzyć z nich bibliotekę *libtekst_wieloliniowy.a*
   - funkcję *main* można umieścić w pliku *main.c* i zapisać w katalogu o nazwie np. *program*, w systemie podkatalogów o strukturze podobnej jak powyżej: *src, lib, include, bin, obj*
   - w katalogu *src* napisanie pliku *Makefile* służącego do budowania kodu
   - wykorzystanie mechanizmu zmiennych środowiskowych (dostępnych w pliku *Makefile* podobnie jak inne symbole: *$(ENV_VAR)* – gdzie *ENV_VAR* jest zmienną środowiskowo zdefiniowaną np. za pomocą (w powłoce bash): *export ENV_VAR*=...)
   - zmienna środowiskowa może np. zawierać nazwę DEBUG lub RELEASE i powodować, że pliki pośrednie i binarne nie są zapisywane bezpośrednio w katalogach *obj* i *bin*, ale w odpowiednich podkatalogach, np.: *obj/RELEASE*, *obj/DEBUG*, *bin/RELEASE*, *bin/DEBUG* (czyli korzystając z przykładowej nazwy powyżej, np. *obj/$(ENV_VAR)*)

Warunki zaliczenia:

1. Obecność na zajęciach i wykonanie co najmniej kroków 1-7
2. Oddanie sprawozdania o treści i formie zgodnej z regulaminem ćwiczeń laboratoryjnych, zawierającego m.in.:
   1. opis wykonanych zadań
   2. kod utworzonych i zmodyfikowanych plików *Makefile*
3. wnioski