VEŽBA 4

Softverski projekti većeg obima retko su sadržani u samo jednom fajlu sa izvornim kodom. Programski jezik C obezbeđuje sve mehanizme potrebne da bi se projekat podelio u manje celine – module – koji se potom prevode odvojeno koristeći isti kompajler (prevodilac) za jezik C. Konačan softver – izvršni fajl, biblioteka, deljena biblioteka ili u slučaju mikrokontrolera binarni (ili hex) paket – dobija se povezivanjem pojedinačno prevedenih modula pomoću softvera poznatom pod imenom **linker** (povezivač).

Takođe, veliki projekti se obično razvijaju tokom dužeg perioda vremena, a u tom procesu učestvuje veći broj ljudi. Na taj način, svako od njih radi na svom zadatku (na programskim modulima na kojima radi) doprinoseći ukupnom rezultatu. Vremenom nastaje veći broj verzija od kojih nisu sve podjednako uspešne – neke su samo bezuspešni eksperimenti, dok ideje koje se pokažu kao dobre trajno ulaze u sastav konačnog softvera. Ovakav razvoj softvera podržan je odgovarajućim softverom od kojih je danas možda najzastupljeniji **git**. U pitanju je distribuirani sistem za upravljanje verzijama softvera (*engl. distributed version control system*).

Moduli u jeziku C

Modul je fajl (sa ekstenzijom `.c`) koji sadrži izvorni kod programa i može sadržati celokupan program. Međutim, program može bit raspodeljen u više ovakvih modula, pri čemu se svaki od njih prevodi potpuno odvojeno od drugih, zbog čega treba da sadrži **sve** informacije koje su za to potrebne. Pre svega je neophodno da sadrži informacije o funkcijama i promenljivama koje će se koristiti u datom modulu, a sadržane su u drugim modulima. Pri tome, ključnu ulogu imaju tzv. **heder** (*engl. header*) fajlovi sa ekstenzijom `.h`.

Primer dva modula

Pretpostavićemo da se program sastoji od dva modula: `main.c` i `worker.c`. Svaki modul (osim možda glavnog, koji sadrži funkciju `main`) bi trebalo da ima i svoj heder fajl, u ovom slučaju `worker.h`. U primeru, modul worker sadrži funkciju `zadatak` koja može biti pozivana iz drugih modula, zbog čega će prototip ove funkcije biti objavljen u hederu tog modula. Taj heder će biti uključen u sve module koji treba da pozivaju funkciju `zadatak`, a radi provere ispravnosti, biće uključen i u modul `worker`. Prvo sledi modul worker:

```
// modul worker.c
#include "worker.h"

int zadatak(char* s, unsigned n)
{
  // ... obrada ce biti preskocena
  return 0;
}
```

Zatim sledi i heder fajl ovog modula – njegov zadatak je da sadrži sve objave o modulu koje treba da budu poznate drugim modulima koji sa ovim modulom treba da komuniciraju. Radi provere ispravnosti (neusaglašenosti sa sadržajem modula i oprečne informacije biće prijavljene kao greške što će dati priliku da se to ispravi):

```
// sledeci makroi sprecavaju visestruko ukljucivanje
#ifndef WORKER_H
#define WORKER_H

// prototip funkcije zadatak
int zadatak(char* s, unsigned n);

#endif
```

Konačno, glavni modul će pozvati funkciju `zadatak`, pa treba da sadrži:

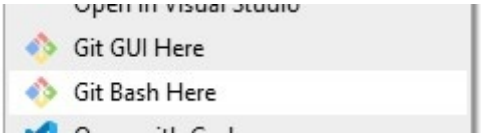
```
// sistemski hederi
#include <stdio.h>

// hederi drugih modula
#include "worker.h"

int main()
{
  int r;
  char m[] = "string za obradu";
  // ...
  // poziv funkcije zadatak
  r = zadatak(m, strlen(m));
  // ...
  return 0;
}
```

Rad sa većim brojem modula u razvojnim okruženjima

Integrisana razvojna okruženja (*engl. Integrated Development Environment – IDE*) podržavaju rad sa više modula preko koncepta **projekata**. Kreiranjem projekta dobija se mogućnost da se projektu dodaju fajlovi – moduli i hederi, a okruženje automatizovano upravlja njihovim prevođenjem (*engl. build*). Ipak, programer treba da zna koje fajlove treba da kreira i doda projektu jer taj deo procesa nije automatizovan.

`git` – alat za upravljanje izvornim kodom

Da bi se koristio alat git, neophodno je kreirati tzv. **reopzitorijum** (*engl. repository*). On uvek ima formu direktorijuma (foldera) koji je posebno opremljen za rad sa git-om. Dva najčešća načina formiranja repozitorijuma je **kreiranje** (novog) ili **kloniranje** (već postojećeg). Za komunikaciju postoji mnogo različitih alata, u ovoj vežbi će se za rad koristiti osnovni alat pozivanjem iz komandne linije (konzole). Pod operativnim sistemom Windows, git konzola se može pozvati pomoću kontekst menija iz FileExplorer-a izborom opcije Git Bash Here nakon što je izabran direktorijum (folder) koji sadrži postojeći (ili budući) repozitorijum:



Izbor ove opcije otvoriće terminal u kojem tekući direktorijum odgovara repozitorijumu i sve

komande koje se zadaju odnosiće se na njega.

Kreiranje novog repozitorijuma

```
git init
git config --local set user.name <ime autora>
git config --local set user.email <e-mail autora>
```

Prva komanda se zadaje unutar direktorijuma gde se nalaze izvorni fajlovi projekta. Tek nakon ove komande, direktorijum postaje repozitorijum i moguće je započinjanje rada na praćenju verzija. Drugi i treći red postavljaju identitet autora koji će se beležiti uz sve akcije nad repozitorijumom.

Trenutni status repozitorijuma

```
git status
```

Daje izveštaj koji su fajlovi izmenjeni ili novi od poslednje komit operacije. Takođe, obaveštava ako su fajlovi menjani od poslednjeg prebacivanja fajlova u praćenje.

Dodavanje fajlova u praćenje (staging area)

```
git add <ime fajla>
```

Komitom će biti oubhvaćeni samo fajlovi koji su predati na praćenje (staging area) gornjom operacijom.

Snimanje verzije – komit

```
git commit -m "Poruka koja se skladisti uz komit"
```

Komit je osnovna jedinica pamćenja git alata. Predstavlja svojevrsnu fotografiju stanja fajlova projekta u trenutku izvođenja gornje operacije. Zapravo, ne svih fajlova u radnom direktorijumu, nego fajlova prebačenih u zonu praćenja (staging area) u trenutku izvođenja operacije komit.

Provera stanja komita

```
git log
```

Ova komanda će dati pregled do sada sačuvanih komita. Ispisuje se autor komita, vreme i poruka sa kojom je komit predat.

Zadaci

1. Po uzoru na primer sa dva modula u prethodnom tekstu, kreirati projekat sa dva modula (`main.c` i `task.c`). Modul task.c treba da implementira funkciju

   ```
   int sum(int a, int b);
   ```

   Ona treba da vraća sumu celih brojeva od `a` do `b`. Pripremiti i odgovarajući heder fajl. Testirati pozivanjem iz modula main.c pri čemu brojeve `a` i `b` za testiranje unosi korisnik preko terminala. Pretpostaviti da je $a \leq b$. Kompajlirati i testirati. Korigovati sve dok ne bude radilo ispravno.
2. Kreirati git repozitorijum i upisati u njega svoje podatke (ime i e-mail, prekidač `--local` čini

da će podaci biti lokalni za repozitorijum). Dodati sve relevantne fajlove u praćenje i napraviti prvi komit. Proveriti status prvo sa `git status`, a potom sa `git log`.

3. Modifikovati funkciju `sum` tako da više nije neophodna pretpostavka o `a≤b`, nego će funkcija uvek dati odgovarajući rezultat u vidu sume brojeva između dve granične vrednosti uključujući i njih obe. Izvršiti izmene, kompajlirati, testirati. Korigovati sve što je potrebno dok testiranje ne pokaže da program radi kao što je očekivano. Napraviti novi komit koji sadrži sve izmene. Proveriti status.

Dodatni neobavezni zadatak:

4. Kreirati `.gitignore` fajl[1] koji će omogućiti da git status više ne prijavljuje fajlove koje generiše Visual studio kao fajlove koje treba predati na praćenje. Dodati i `.gitignore` fajl u sledeći komit.

1 Fajl `.gitignore` je običan tekstualni fajl koji u odvojenim redovima sadrži imena svih fajlova i direktorijuma za koje želimo da ih alat git ignoriše, odnosno ne uzima ih u obzir pri proceni šta još treba dodati u praćenje (pri zadavanju komande `git status`). Uobičajeni džoker karakteri (`*`, `?`) su dozvoljeni.