PP Predmetni projekat
Dokumentacija

Osnovni podaci

Broj indeksa	SW-38/2018
Ime i prezime	Marko Njegomir
Šifra zadatka	PP-IBVCA
Kontrolna tačka	Dodatni zadaci – Novi jezik
Operativni sistem	Ubuntu 20.04.2 LTS
gcc verzija	gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0

Detalji implementacije

U ovoj dokumentaciji je opsian novi jezik koji sam napisao. Ovo je potpuno drugačiji kod od onoga koji sam predao za kontrolnu tačku.

U kodu koji sam predao za kontrolnu tačku se nalaze I moja preostala dva dodatna zadatka, a to su ugnježdeni pozivi funkcija, I ugnježdeni check iskazi. O njima neću ovde pisati zato što sam ih već opisao u dokumentaciji za kontrolnu tačku. Takođe neću poslati ni taj kod, jer sam ga već predao za kontrolnu tačku.

Dalje sledi samo opis novog jezika I detalja implementacije istog.

Motivacija za novi jezik

U početku sam razmišljao da to napišem jezik koji bi ličio na C jezik, samo što bi bio na ćirilici. To bi dovelo I do promene sintakse I gramatike. Ipak mi to nije delovalo previše zanimljivo, zbog toga što bi taj jezik bio previše sličan C jeziku.

To me je navelo da razmislim o tome za šta se sve može definisati I koristiti jezik. Da li moram da napravim jezik koji ću unapred da unesem, a računar da zatim prođe kroz ceo fajl i generiše kod? Ne, jer mogu da kucam kod I dobijam povratnu informaciju već prilikom unošenja koda. Da li može da se napravi netrivijalan jezik koji nema sintaksne greške? U programu sam uspeo da pokrijem sve slučajeve gde bi mogla nastati sintaksna greška. Da li mogu sam da definišem token koji bi označavao kraj parsiranja? Može, I tako dajem opciju korisniku da sam završi parsiranje u bilo kojem momentu. Da li je bitan broj linija koda? Da li moram poštovati cdecl konvenciju prilikom

generisanja koda? Da li mogu slobodno da koristim sve dostupne registre? Na ova, ali i mnoga druga pitanja, odgovor je u jeziku i kodu koji sam napisao, a koji ću ovde pokušati da objasnim.

Jezik za indukciju eliminacijom

Moja ideja je bila da napravim novi jezik za indukciju eliminacijom. Indukcija eliminacijom se lepo može objasniti na primeru postavljanja dijagnoze pacijenta. Tu doktor postavlja pitanja, sprovodi različite analize i pri tom mu dobijene informacije omogućavju da eleminiše potencijalne bolesti. Taj postupak se ponavlja sve dok ne preostane alternativa, i može se zaključiti koja je bolest u pitanju.

Najzanimljiviji primer za takvu vrstu indukcije je igra iz skagalice zvana Skočko (eng. *Mastermind*). Tu igrač zadaje upit u vidu kombinacije u kojoj se na svakoj od četri pozicije može pojaviti jedan od 6 znakova. Cilj je da se pogodi tražena kombinacija koja je na startu igre nepoznata. Verovatnoća da se u prvom pokušaju pogodi kombinacija je 1/1296. Nakon unosa svake kombinacije, igrač dobija povratnu vrednost od računara, gde mu je saopšteno koji broj znakova u kombinaciji je na pravom mestu (u vidu crvenih loptica), i koji broj znakova iz kombinacije postoji i u traženoj kombinaciji, ali nisu na pravom mestu (u vidu žutih loptica). Nakon svakog pokušaja broj mogućih kombinacija se smanjuje. Kada se nakon serije upita i odgovora broj mogućih kombinacija svede na jednu jedinu, onda se sa sigurnošću može saopštiti tražena kombinacija.

Igrač može da pogodi i kombinaciju na sreću i pre nego što je sveo broj mogućih kombinacija na jednu. To se retko dešava, pa je dobra strategija da igra da u svakom koraku svojim upitima eliminiše što više mogućih kombinacija, tako da čak i u najgorem slučaju kada nema sreće, on bude u stanju da pogodi traženu kombinaciju u 5 koraka (ako koristi strategiju Donalda Knuta).

Leksika

Jezik podržava unos reči na ćirilici, kao I brojeve.

Za ostale unose se prijavljuju greške koje se ispisuju na ekranu, ali se izvršavanje programa ne zaustavlja.

Teškoća prilikom implementacije je bila u tome što alat Flex parsira unos jedan po jedan bajt. UTF-8 karakteri se sastoje od 3 bajta. Zbog ovoga je bilo potrebno pisati regularne izraze na pomalo nekonvencionalan način.

Prvi primer je da se nije mogao navesti opseg unutar uglastih zagrada [а-шА-Ш]. Ako se UTF-8 karakter navodi unutar takvih zagrada, onda je potrebno staviti ga unutar duplih navodnika „“, da bi ih Flex tretirao kao jednu reč od 3 karaktera, jer oni se sastoje od 3 bajta. Zbog toga sam morao eksplicitno da navodim sve karaktere unutar uglastih zagrada.

Podržan je rad sa velikim i malim slovima. U tim pravilima nisam morao da koristim uglaste zagrade, pa sam mogao da stavljam karaktere i bez duplih navodnika.

Reči koje su podržane su znakovi na ćirilici, brojevi koji će se kasnije tumačiti kao znakovi, dve komande za početak igre, i komanda za kraj igre.

Svi prazni prostori, tabovi i novi redovi se ignorisu.

Ideje za proširivanje leksike

Leksika se može proširiti novim komandama za pokretanje snimljenih partija. Takože se mogu dodati reči koje bi omogućile korisniku da unese više znakova odjednom, kao u primeru „četri puta skočko“.

Gramatika

U gramatici postoje tokeni za

- Novu igru
- Kraj
- Znak
- Broj.

Ideja je bila da se stvori takva gramatika koja omogućava indukciju eliminacijom na primeru igre skočko, takva da ne dolazi do sintaksnih grešaka, i koja omogućava igraču da ako želi, igra beskonačno mnogo partija za redom, sa opcijom da prekine u svakom momentu. Generalno je ceo tok interakcije obogaćen vizualnim prikazem table i poruka u boji.

Svi tokeni za znakove pre početka igre uzrokuju ispis poruke upozorenja da se mora započeti igra. Znak za kraj u svakom momentu može da prekine igru. Tada se završava upis generisanog koda u fajl, ali samo ako je igra već prethodno bila započeta.

Čekanje da se eksplicitno unese token za početak igre je uveden da bi korisnik morao da prebaci pismo na ćirilicu, jer posle ipak može da koristi brojeve umesto ćiriličnih naziva za znakove.

Token za novu igru pokreće funckiju za generisanje nove igre. To obuhvata resetovanje svih podešavanja, generisanje nove tražene kombinacije nasumičnim izborom i finalizovanje generisanja koda prethodne partije, ukoliko ta prethodna partija nije propisno završena. Takođe se generiše i kod za početak nove igre u novom fajlu.

Pravilo za igru je rekurzivno, da bi se omogućilo igranje više uzastopnih partija.

Lista kombinacija može biti prazna zbog toga što korisnik može u bilo kom momentu započeti novu igru, ili prekinuti program unosom tokena za kraj.

Kombinacija sadrži rekurzivno pravilo koje omogućava da se unese proizvoljan broj znakova za redom. Svako od tih unosa za znak može biti ili token znak ili broj u opsegu od 1 do 6, koji odgovaraju znacima. Ovo pravilo za unos za znak ima povratnu vrednost koju koristi pravilo kombinacija, da bi znalo koji je znak unet.

U pravilu kombinacija se za svaki unos znaka poziva funkcija unesi_znak. Ta funkcija ignoriše neispravne unose koji mogu nastati unošenjem brojeva van opsega od 1 do 6. Ako je igra završena, neće biti dodati novi znakovi.

Kada je unos ispravan i kada igra još uvek traje, znak će biti zapamćen u trenutnoj kombinaciji. Ako su očitana 4 znaka, onda će se očistiti terminal, i biće pozvana funkcija za odigravanje kobminacije. U toj funkciji će se generisati znakovi potrebni za ispis na osnovu evaluacije trenutnog pokušaja. Tu evaluaciju vrši funkcija evaluiraj poziciju, koja pravi histogram znakova odigranih u kombinaciji, i dolazi do informacija o ispravnosti unete kombinacije u poređenju sa traženom kombinacijom. Na kraju funkcije za odigravanje kombinacije se ispisuje nova tabla sa odigranom kombinacijom i indikatorima tačnosti u vidu crvenih i žutih loptica.

Nakon odigrane kombinacije se poziva funkcija za ispis nakon unete kombinacije. Ova funkcija generiše kod za unetu kombinaciju, i ako je ta kombinacija ista kao i tražena, onda se ispisuje i poruka o kraju partije, i postavlja se stanje partije u završena. Ako je igra završena biće i generisan poslednji deo koda neophodan za ponovno pokretanje te odigrane partije.

Ostale pomoćne funkcije su trivijalne i sam naziv je dovoljan da se zaključi koji posao obavljaju.

Ideje za proširivanje gramatike

Gramatika bi se mogla proširiti da podrži unos više znakova komandom tipa „4 puta skočko“. Takođe bi se mogla proširiti da podrži pokretanje ponovljenog snimka neke od snimljenih partija za koje je generisan kod.

Generisanje koda

Kod koji je generisan je x86 assemblerski kod (i386). U pitanju je AT&T sintaksa.
Ja sam se odlučio za 32-bitnu verziju koda, pa je zbog toga potrebno postaviti opciju -m32 prilikom ručnog kompajliranja.

Ideja koda je da se generiše igra skočko koja će odigrati poteze koje je korisnik odigrao u snimljenoj partiji.

Jedna kombinacija je smeštena u jedan registar. To je postignuto enkodovanjem znakova u binarni oblik, gde svaka pozicija bita označava jedan znak. Pošto je registar 32-bitni on može da se podeli na 4 dela od po 8 bita, od koji svaki ima postavljen bit za po jedan znak u kombinaciji.

Sve operacije se onda nakon učitavanja mogu sprovesti upotrebom maski i rotiranjem, tj shiftovanjem. Tako se izbegava nepotreban pristup memoriji koji može biti sporiji od upotrebe nekoliko dodatnih operacija sa vrednostima koje su već u registru.

Da bi se postigao efikasan rad sa registrima, ne poštuje se cdecl konvencija pozivanja funkcija. To znači da se ne moraju čuvati vrednosti registara na početku funkcije, i ne moraju se učitavati u registre prilikom povratka. To nepoštovanje cdecl konvencije oslobađa i dosta novih registara za upotrebu, koji se koriste za smeštanje maski. Generalno, funkcije vraćaju povratne vrednosti u registrima.

Osim data sekcije i predefinisanog koda za funkcije, i kraj programa, generiše se i kod za ispis unete kombinacije i stanja table. To je deo koji se generiše nakon unosa kombinacije. Pre ispisa se očisti terminal. Nakon ispisa postoji mala pauza da bi se mogao ispratiti tok igre.

Ideje za proširivanje koda

U asemblerskom kodu sam već napisao funkciju koja računa broj preostalih kombinacija. Taj broj bi se mogao prikazati. Takođe bi se mogao prikazati i broj bodova koji se mogu osvojiti zasnovan na novim pravilima koja sam osmislio za igru skočko, a koja ne kažnjavaju dobre igrače koji nisu imali sreće da pogode kombinaciju u prvih par koraka.

Fajlovi od 0 bajtova i 0 linija koda

Ovi fajlovi imaju sav kod enkodovan u nazivu fajla, u vidu hexadecimalnih karaktera. Veličina naziva svakog fajla je 150 karaktera zbog ograničenja operativnih sistema. Zbog toga se pravi više fajlova od 0 bajtova prilikom enkodovanja.

Dekodovanje se sastoji od iščitavanja naziva fajlova, i pretvaranja nazad u binarni fajl. Fajl koji se enkoduje i dekoduje može biti bilo kog formata. Ovde sam se odlučio da enkodujem dobijeni asemblerski fajl koji se dobija na kraju partije.

Kod za konverziju je napisan u c++ zbog jednostavnijeg rada sa stringovima u odnosu na jezik c.

Make alat

U make alat sam dodao novu komadnu za dekodovanje zero byte fajlova i puštanje snimka odigrane partije.

Način pokretanja je sledeći

- make clean

- make
- ./скочко
- make replay REPLAY=igra1

Dostupan sam za sva ostala pitanja i rado ću odgovoriti na njih. Nisam imao vremena da sve pokrijem u dokumentaciji.