Prevođenje programskih jezika

Na Desktopu napraviti direktorijum sa nazivom **PPJ.jan2.ime.prezime.brojIndeksa.godina** i u njemu sačuvati svoj rad. Dakle, ako **Pera Perić** sa indeksom **123/2015** polaže ispit potrebno je da na Desktopu napravi direktorijum sa nazivom **PPJ.jan2.Pera.Peric.123.2015** i u njemu da sačuva svoj rad. **Sintaksno neispravni zadaci se ne pregledaju. Vreme za izradu ispita je 3h. Uslov za polaganje ispita je minimum 20p.**

Sintaksna analiza naviše

Naše mlađe kolege se žale kako su im dinamičke strukture podataka na Programiranju 2 teške i zamolile su nas da im pomognemo da savladaju povezane liste. Potrebno je da napišemo interpreter koji će im omogućiti sledeće:

1. Interpreter treba da omogući jednostavno definisanje liste, prikazivanje elemenata u listi i operaciju dodele. Liste se mogu definisati nad brojevima ili nad stringovima. Lista ne može istovremeno sadržati i brojeve i stringove. Lista može biti proizvoljne dužine. Funkcija *print* štampa elemente liste unapred, a funkcije *print_r* štampa elemente liste unazad. U slučaju da se greška dogodi, obavestiti korisnika i nastaviti dalje sa izvršavanjem.

```
list<number> x;                          // definiše praznu listu
list<number> y = {1,2,3,4,5,6};
list<string> z = {“a”, “b”, “c”};
y = z;                                   // nije moguće izvršiti dodelu
x.print();                               []
z.print_r();                             [“c”, “b”, “a”]
list<string> x = {“a”, “b”, “c”};        // x je već definisano
```

2. Interpreter treba da omogući sve uobičajene dinamičke operacije nad listama. Funkcija *push_back*(x) dodaje element *x* na kraj liste, funkcija *push_front(x)* dodaje element *x* na početak, a funkcija *push(x,k)* dodaje element *x* na poziciju *k* u listi. Ekvivalentno, funkcije *pop_back()*, *pop_front()* i *pop(k)* redom uklanjaju element sa kraja, sa početka i sa pozicije *k* u listi. Nije moguće dodavati brojeve u listu stringova i obrnuto. Operacija *find(x)* ispituje da li se x nalazi u listi, a operacija *get(k)* vraća element na poziciji *k* u listi. Operacije se mogu proizvoljno nadovezivati.

```
x.push_back(5).push_front(10).push(3,1);
x.print();                                   [5,3,10]
y.get(2).print_r();                          [3]
z.find(“a“);                                 [“a”]
z.find(“k”);                                 []
z.get(10).print_r();                         []
```

3. Interpreter treba da omogući osnovne aritmetičke operacije sa sledećim značenjem: + - unija dve liste, * - presek dve liste, - - razlika dve liste, unarni minus obrtanje liste, () – grupisanje. Na rezultat izraza mogu se primeniti sve do sada uvedene operacije. Ako se operacije primene na liste različitih tipova, tada se umesto brojeva u izrazu koriste njihove string reprezentacije.

```
list<number> k = {1,2,3};
list<string> s = {“d”, “e”};
-((k+s).push_back(“f”)*(s-k).push_back(“1”)).push_front(“11”);   [“11”,“1”, “d”, “e”]
```

4. Interpreter treba da omogući upoređivanje listi (<, <=, >, >=, ==, !=). Liste se upoređuju kao stringovi, tj. leksikografski.

```
x + y == -y*x;                                  False
(x+y).push_back(1) < y.push_front(7).pop_back();    False
```

Napomena: Makefile je obavezan deo rešenja.

Sintaksna analiza naniže

Uz pomoć rekurzivnog spusta implementirati sintaksni analizator koji omogućava proveru sintaksne ispravnosti dela pod 1 iz prethodnog zadatka. Sintaksni analizator ne treba da razlikuje tipove podataka prilikom dodele. U komentaru programa je neophodno ostaviti sređenu gramatiku i skupove izbora.

```
list<number> x;
list<number> y = {1,2,3,4,5,6};
list<string> z = {“a”, “b”, “c”};
y = z;
x.print();
z.print_r();
list<string> x = {“a”, “b”, “c”};
```

Ispit traje 3h.