Programske paradigme

Imperativna paradigma

Milena Vujošević Janičić

www.matf.bg.ac.rs/~milena

Pregled

Pregled

Imperativna paradigma

- Imepprativna paradigma je prvonastala programska paradigma.
- Nastala je pod uticajem Fon Nojmanove arhitekture računara (tj. ova arhitektura nameće ovaj stil programiranja, prilagođen mašini, a ne čoveku).
- U rešavanju problema prednost se daje algoritmima pa se imperativno programiranje naziva i Algoritmaski orijentisano programiranje
- Podaci i algoritmi postoje nezavisno: Podaci+Algoritmi=Programi

Imperativna paradigma

- Kao što se u govornom jeziku zapovedni način (ili imperativ) koristi za izražavanje naredbi, tako se imperativni programi mogu posmatrati kao niz naredbi koje računar treba da izvrši.
- Pored naredbi, ključan je i redosled izvršavanja naredbi — procedura.
- Imperativna paradigma i proceduralna paradigma se često koriste kao sinonimi.
- Imperativno programiranje karakteriše izračunavanje u terminima naredbi koje menjaju *stanje* programa.

Stanje programa

- Stanje programa čine sve sačuvane informacije, u datom trenutku vremena, kojima program ima pristup (preko memorije).
- Izvršavanjem programa generiše se niz stanja. Prelaz iz jednog stanja u sledeće je određen komandama koje se izvršavaju.
- Svaki imperativni jezik obezbeđuje raznovrsne komande za modifikaciju stanja (manipulisanje) memorije.

Naredba dodele

- Osnovna komanda za modifikaciju stanja memorije je naredba dodele.
- Naredba dodele vrši povezivanje imena i neke vrednosti, odnosno upisivanje konkretne binarne reči na odgovarajuću memorijsku lokaciju.
- Analogija između memorijskih i (procedurlno) jezičkih elemenata:

Memorija	binarna reč	mem. registar	adresa
Jezik	vrednost	promenljiva	ime

Promenljive i naredba dodele

- Deklaracijom promenljive u proceduralnom jeziku određuje se veličina memorijslog prostora za zapis promenljive. Na primer:

```
Pascal:      var name : Type;
C :          type name;
```

- Promenljiva se povezuje sa nekom vrednošću preko izraza, što se različito opisuje u različitim jezicima. Na primer:

```
Pascal :     V := E
C :          V = E
APL :        V <-- E
```

Naredba dodele i redosled vezivanja

- Redosled povezivanja imena i vrednosti (dodeljivanje) utiče na vrednosti izracunavanja

```
a = 5; b = 3; c = 7;
b = a+1;
c = a+b; /*Redosled izvrsavanja ovih naredbi je bitan!*/
```

- Bitan je i redosled povezivanja (asocijativnost, prioriteti).

```
b = a+b*c/2;
```

Kontrola toka

- Kontrola toka se ostvaruje kroz različite instrukcije i apstrakcije.
- Najveći broj konstrukcija u imperativnim jezicima je odraz hardverske implementacije.
- Imperativna paradigma prolazi kroz različite faze razvoja, a svaku fazu karakteriše viši nivo apstrakcije u odnosu na arhitekturu računara, i udaljavanje od asemblerskih jezika.

Faze razvoja imperativne (proceduralne) paradigme

- Neki autori svaku fazu imperativne paradigme izdvajaju u posebnu programsku paradigmu, a neki ih tretiraju kao potparadigme imperativne paradigme.
  - Operaciona (pod)paradigma
  - Strukturna (pod)paradigma
  - Proceduralna (pod)paradigma
  - Modularna (pod)paradigma

Operaciona (pod)paradigma

- Prva faza programiranja.
- Programiranje zasnovano na dosetkama, trikovima, naziva se i *trik programiranje*.
- Programi su pisani bez opštih pravila, korišćenje su specifičnosti u radu računara, trikovi za uštedu memorije, pisanje samomodifikujućih programa.
- Ogromna produkcija softvera karajem 60-tih godina prošlog veka.
- Jezici: FORTRAN, ALGOL, COBOL, BASIC, ...

Kontrola toka

- Minimalan skup komandi (koji je često korišćen):

```
komanda  ::=
          identifikator = izraz |              (naredba dodele)
          komanda; komanda |                   (sekvenca)
          labela : komanda |                   (obelezavanje)
          GOTO labela |                        (naredba skoka)
          IF (log_izraz) [THEN] GOTO labela    (selekcija)
```

Kontrola toka

- Ove upravljačke strukture su nastale kao odraz (analogoni) struktura u programu na mašinskom jeziku.
- Prema Fon Nojmanovom konceptu računara, instrukcije slede jedna za drugom (sekvenca naredbi u imperativnom jeziku), a redosled se može promeniti korišćenjem GOTO (Jump) – naredbe.
- U imperativnim jezicima javljaju se 2 oblika GOTO-naredbe (u IF-naredbi i bez IF-naredbe).
- Korišćenje prethodnih naredbi dovodi do pisanja nepreglednih programa, teških za modifikaciju ("Špageti-programi").
- To je posledica prilagođavanja jezika mašini, a ne čoveku.

Špageti programiranje — primer u C-u

```
#include <stdio.h>

main()
{
 float x,y;
 int    n;
   n=1;
l20: scanf("%f", &x);
   y=x;
   if(x<0) goto l30;
   y=-x;
l30: printf("y = %f\n",y);
   n=n+1;
   if (n!=5) goto l20;
}
```

Špageti programiranje - primer u Pascal-u

```
program idina;

  label 20, 30;
  var x,y:real;
      n:integer;
begin
    n:=1;
20: readln(x);
    y:=x;
    if x<0 then goto 30;
    y:=-x;
30: writeln(y);
    n:=n+1;
    if n<>5 then goto 20
end.
```

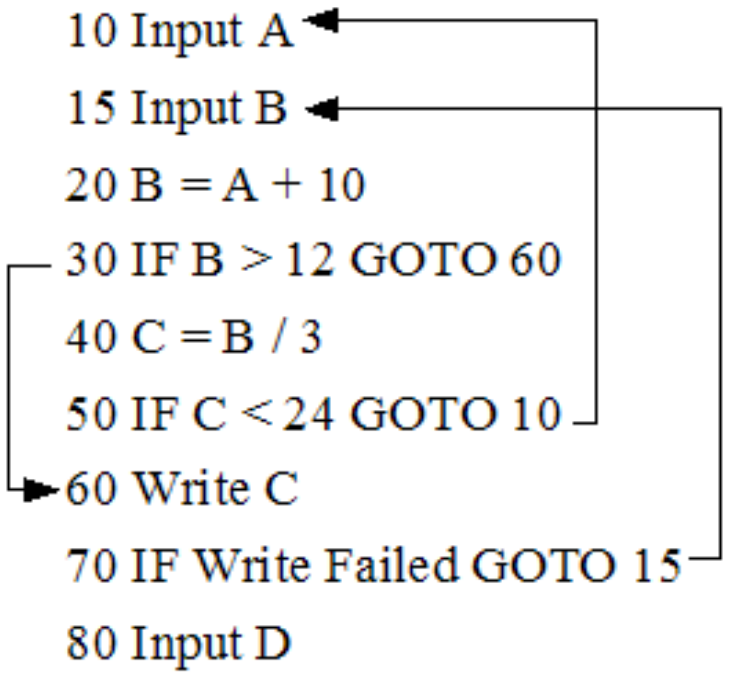
Špageti programiranje — primer u C-u

```
main() {
   int i,n;
   float h, x0, x, y;
   printf ("Unsete h, x0 i n\n");
   scanf("%f%f%d",&h,&x0,&n);
    if (n<0) goto komentar;
   printf("Pocetak \n");
   i=0;
poc: x=x0+i*h;
   y=x*x;
   printf("x=%f,  y=%f\n", x, y);
   if(i<n) goto uvecanje;
   return 0;
uvecanje: i=i+1; goto poc;
komentar: printf("Nekorektno zadato n\n");
 }
```

Špageti programiranje - primer u Pascal-u

```
program sagoto;
   label poc, kraj;
   var i,n: integer;
       h, x0, x, y: real;
begin
    writeln ('Unsete h, x0 i n');
    readln (h, x0, n);
    writeln ('Pocetak');
    i:=0;
poc: x:=x0+i*h;
    y:=x*x;
   writeln('x = ', x, '  y = ', y);
   if(i>=n) then goto kraj;
   i:=i+1;
   goto poc;
kraj:
  end.
```

Špageti programiranje — primer



Operaciona paradigma — problemi

- Teška čitljivost programa i održavanje programa.
- Kreirani softver je nepogodan za izmene i prilagođavanje novim situacijama.
- Softverska kriza 1970-ih — zbog loše prakse programiranja softver nije mogao da dostigne mogućnosti hardvera.
- Uzrok: nekontrolisana upotreba GOTO-naredbe.
- Rešenje: strukturno programiranje — disciplinovan pristupu programiranju, bez nekontrolisanih skokova i uz korišćenje samo malog broja naredbi za kontrolu toka programa.

Strukturna paradigma

- C. Bohm i G. Jacopini su 1966. g. publikovali naučni rad u kojem su dokazali da se svaki prost program može izraziti pomoću 3 upravljačke strukture:
  - sekvenca (naredba za naredbom),
  - selekcija (odluka da li da se izvrši neka naradba zavisno od tačnosti ili netačnosti nekog uslova)
  - ponavljanje (ponavljanje bloka koda vraćanjem na početak sve dok je ispunjen neki uslov).
- Kasnije su usledeli radovi: Dijkstra-e, Knuth-a i E. Ashcroft i Z. Manna . . . u kojima je pokazano da goto naredba nije neophodna.
- Dijkstra je napisao 1968. čuveno pismo "Go To Statement Considered Harmful".

Strukturna paradigma

- Strukturno programiranje nastaje kao nastojanje da zapis programa bude pregledniji.
- Akcenat je na programskim strukturama u kojima svaka komanda ima jednu ulaznu i jednu izlaznu tačku.
- Cilj je da se proceduralni jezik više prilagodi čoveku.
- Minimalan skup upravljačkih struktura čine:

```
komanda :: =
        identifikator := izraz          |   (naredba dodele)
        komanda ; komanda               |   (sekvenca)
        IF  log_izraz   THEN-grana
                        ELSE-grana      |   (selekcija)
        WHILE  log_izraz DO naredba     |   (iteracija)
```

Strukturna paradigma

- Zbog preglednijeg zapisa programa, najčešće se uvode dodatne upravljačke strukture, kao što su:

  `CASE (switch)`
  `FOR`
  `REPET-UNTIL (do-while)`

- Kontrolne naredbe su sintaksičke strukture preko kojih se definiše redosled u kojem se vrši dodeljivanje.
- Sve kontrolne strukture mogu da se svrstaju u 3 kategorije: sekvencijana kompozicija, alternacija (selekcija) i iteracija.

Strukturna paradigma

- Programi zapisani pomoću ovih upravljačkih struktura su pregledniji, jasniji i često kraći.

```
#include <stdio.h>
main()
{
 float x,y;
 int    n;
   n=1;
l20: scanf("%f", &x);
   y=x;
   if(x<0) goto l30;
   y=-x;
l30: printf("y = %f\n",y);
   n=n+1;
   if (n!=5) goto l20;
}
```

```
#include <stdio.h>
main()
{
 float x,y;
 int    n;
 printf("Unesite 4 realne vrednosti\n");
 for(n=1; n<5; n++)
  {
    scanf("%f", &x);
    y=x;
    if(x>=0) y=-x;
    printf("y = %f\n",y);
  }
}
```

Strukturna paradigma

```
program idina;
  label 20, 30;
  var x,y:real;
      n:integer;
begin
    n:=1;
20: readln(x);
    y:=x;
    if x<0 then goto 30;
    y:=-x;
30: writeln(y);
    n:=n+1;
    if n<>5 then goto 20
end.
```

```
program bezIdina;
  var x,y:real;
      n:integer;
begin
    writeln('Unesite 4 realne vrednosti');
    for n:=1 to 4 do
    begin
      readln(x);
      y:=x;
      if x>=0 then y:=-x;
      writeln('y = ', y);
    end
end.
```

Proceduralna paradigma

- Apstrakcija kontrole toka — podrutine (funkcije, procedure, metodi, korutine, potprogrami).
- Podrutine predstavljaju apstrakciju niza naredbi.
- Poziv podrutine je poziv na apstarkciju, približava programiranje deklarativnosti.
- Podrutina izvršava svoje operacije u ime svog pozivaoca.
- Nastanak potprograma prethodi strukturnom programiranju.

Proceduralna paradigma+strukturno programiranje

- Svaki podrutina ima svoje lokalne podatke i algoritam.
- Svaki podrutina je nezavisna od ostalih.
- Razvija se mehanizam prenosa parametara.
- Nastaju korisnički definisani tipovi.
- Uvodi se vidljivost i doseg podataka.
- Podržava se ugnježdavanje podrutina.
- Podrutine su osnovni blokovi za podržavanje modularnog programiranja.

Proceduralna paradigma

- U imperativnim programskim jezicima, podrutine su najčešće procedure i funkcije.
- Procedure — nemaju povratnu vrednost.
- Funkcije — imaju povratnu vrednost.
- Jednim imenom se procedure i funkcije nazivaju potprogrami.
- Sintaksa potprograma je obično jednostavna:

```
Definicija potprograma:
                    ime( parametar-lista ) { telo }
Poziv potprograma:
                    ime( argument-lista )
```

Potprogrami

- Parametri se često nazivaju formalni parametri, a argumenti, aktuelni parametri.
- Sintaksa parametar-liste, tj. argument-liste je različita u različitim programskim jezicima (obično lako shvatljiva).
- Mogu se razlikovati
  - vrednosni parametri (in-parametri)
  - promenljivi parametri (out-parametri)

Prenos parametara

- Vrednosni parametri služe samo za unos vrednosti u proceduru, mogu se menjati u proceduri, ali ne mogu izneti izmenjenie vrednosti.
- Promenljivi parametri mogu uneti vrednost u proceduru, mogu biti menjani u proceduri i (najvažnije) zadržavaju izmenjene vrednosti po izlasku iz procedure.
- Kao stvarni argumenti vrednosnih parametara obično se pojavljuju izrazi, a kao stvarni argumenti promenljivih parametra obično su promenljive ili pokazivači.

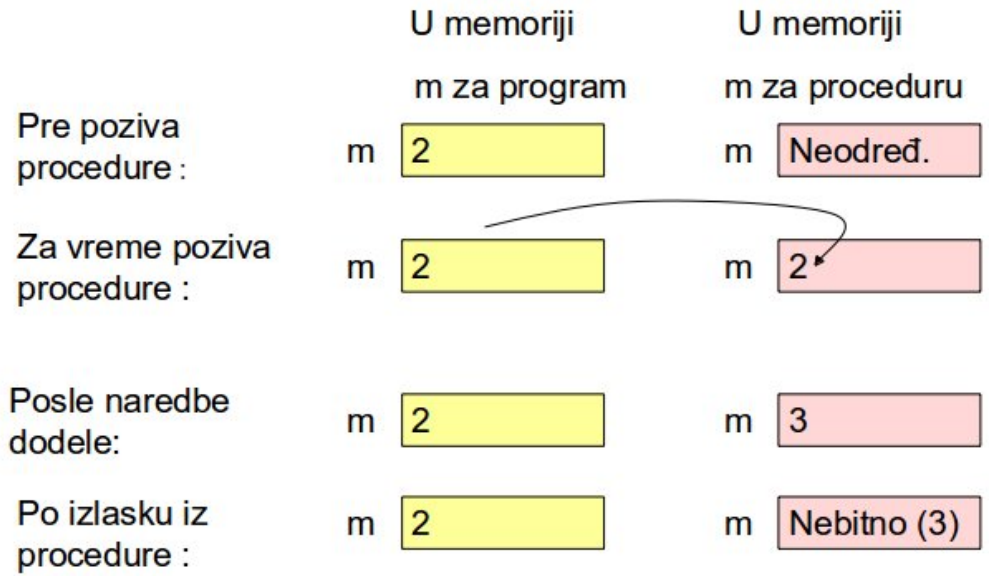
Prenos parametra — preciznije

- Neki jezici imaju samo jedan način prenosa parametara, dok neki dozvoljavaju više načina.
- Osnovne vrste prenosa parametara:
  1. Prenos po vrednosti — argument se evaluira i kopira njegova vrednost u podrutinu (podrazumevan prenos u jezicima nakon Algol 60, Pascal, Delphi, Simula, Modula, Oberon, Ada, C, C++...)
  2. Prenos po referenci — prenosi se referenca na argument, obično adresa (moguće odabrati u svim prethodno navedenim jezicima)

Prenos parametara po vrednosti

```
var m: integer
....
  m:= 2;
  prva (m);
....
```

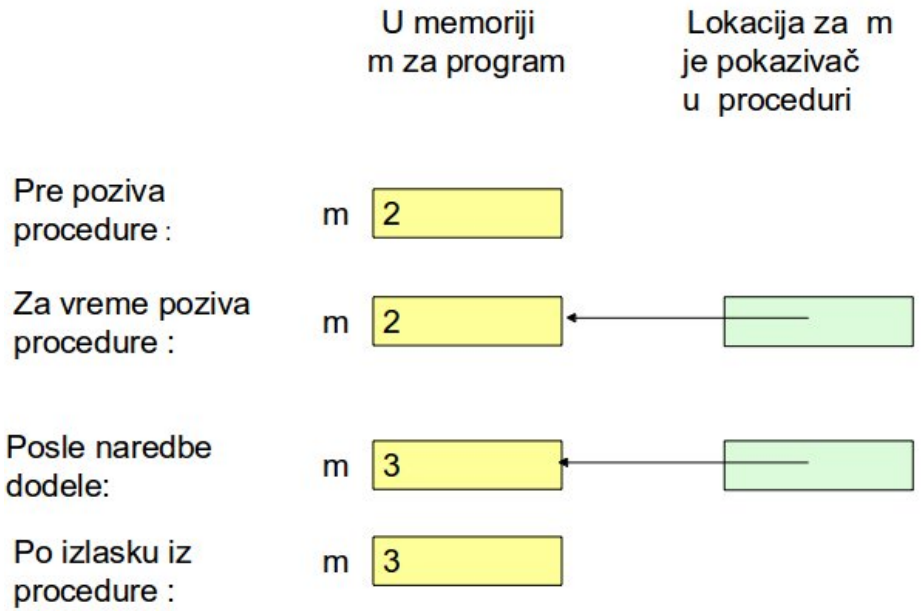
```
procedure prva (m: integer);
        ....
        m:= m+1;
        ....
```



Prenos parametara po referenci

```
var m: integer
....
  m:= 2;
  prva (m);
....
```

```
procedure prva (var m: integer);
        ....
        m:= m+1;
        ....
```



Prenos parametra — preciznije

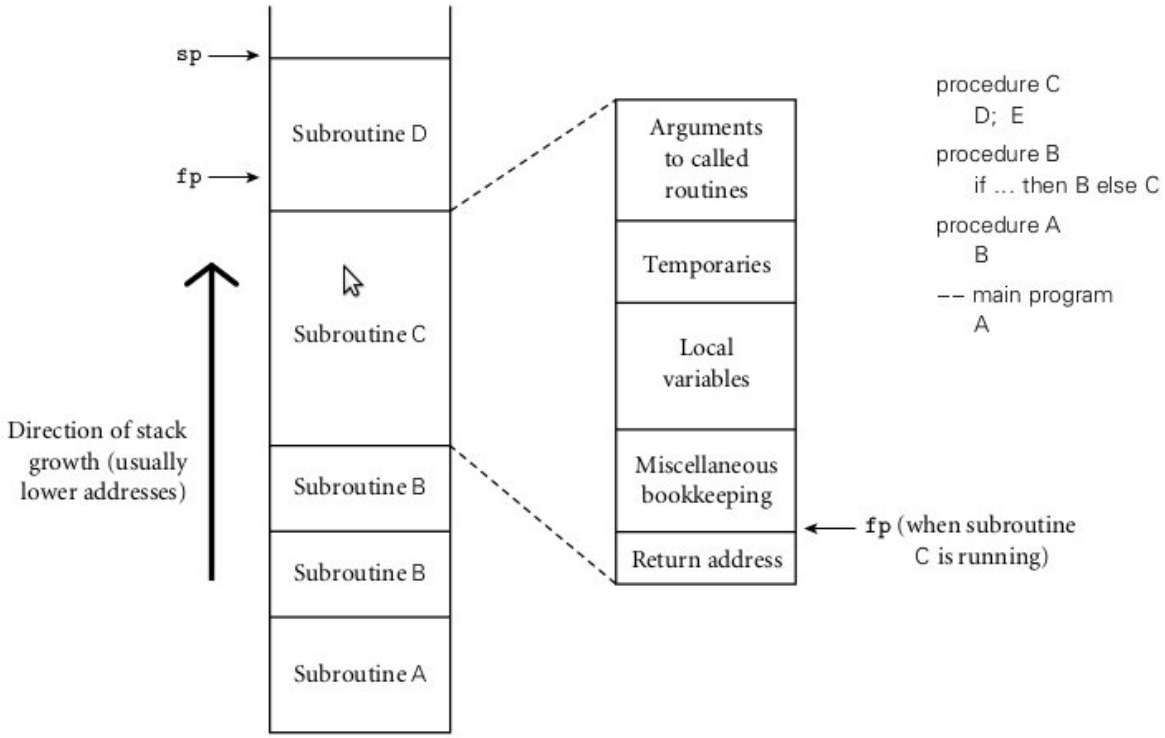
Dodatni načini prenosa parametara:

3. Prenos po rezultatu — na izlasku iz podrutine, vrednost parametra se kopira (prenosi) pozivajućoj rutini (Ada OUT parametri)
4. Prenos po vrednosti i rezultatu — vrednost parametra se kopira na ulasku i na izlasku iz podrutine (Algol)
5. Prenos po imenu — kao u makroima, parametri se zamenjuju sa neevaluiranim izrazima (Algol, Scala)
6. Prenos po konstanoj vrednosti — isto kao kod prenosa po vrednosti, osim što se parametar tretira kao konstanta (npr kvalifikator const u C i C++)

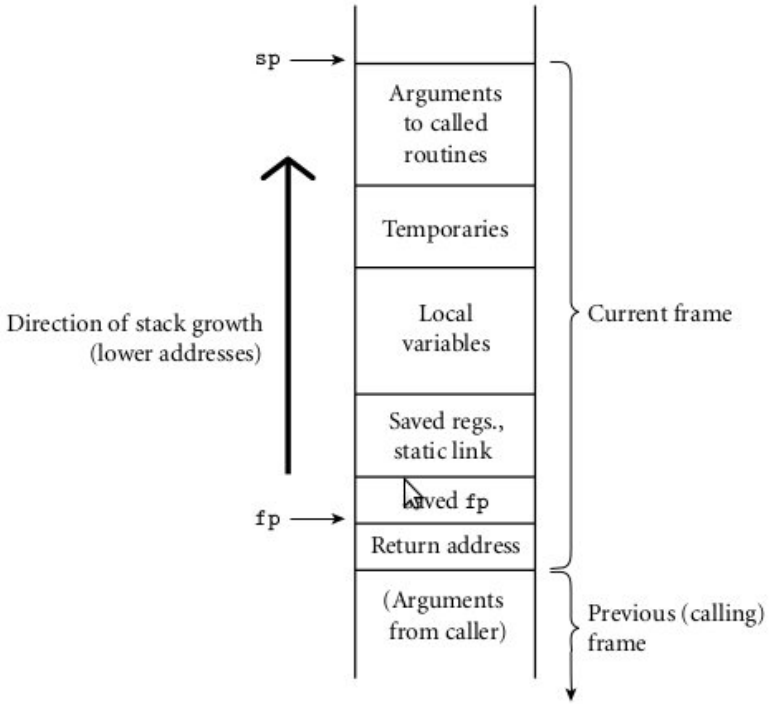
Organizacija memorije

- U većini jezika, prostor potreban za smeštanje podataka potrebnih za izvršavanje potprograma (promenljive, argumenti i slično) se rezerviše na steku.
- Uvođenje steka je imalo za ideju uštedu memorije — u memoriji su podaci samo za trenutno aktivne potprograme.
- Alternativa bi bila da se za svaki potprogram unapred rezerviše potrebna memorija. Ovo onemogućava korišćenje rekurzija i zauzima više prostora nego što je potrebno.
- S druge strane, kreiranje stek okvira povećava cenu poziva potprograma.

Stek okviri


sp
fp
Subroutine D
Subroutine C
Subroutine B
Subroutine B
Subroutine A
Direction of stack growth (usually lower addresses)
Arguments to called routines
Temporaries
Local variables
Miscellaneous bookkeeping
Return address
fp (when subroutine C is running)
procedure C
D; E
procedure B
if ... then B else C
procedure A
B
-- main program
A

Sadržaj stek okvira



Korutine

- Završetak rada potprograma vraća nas u prethodni stek okvir (stek okvir pozivaoca potprograma).
- Pored procedura i funkcija, u imperativnim jezicima mogu se javiti i druge vrste potprograma (upravljačkih struktura), kao što su korutine.
- Korutine omogućavaju „preskakanje" stek okvira i povratak u određeni stek okvir koji nije nužno stek okvir pozivaoca potprograma.

Korutine

- Upotreba može da bude za brz i efikasan izlaz iz duboke rekurzije.
- Slični mehanizmi koriste se u npr objektno orijentisanim programskim jezicima za rad sa izuzecima.
- U C-u korutine nisu podržane direktno u jeziku, već se mogu koristiti funkcije `setjmp` i `longjmp` (iz zaglavlja `setjmp.h`). Ispravna upotreba korutina zahteva precizno poznavanje njihovog mehanizama funkcionisanja.

Modularna paradigma — rane 1970.

- Modularnost podrazumeva razbijanje većeg problema na nezavisne celine.
- Celine sadrže definicije srodnih podataka i funkcija.
- Često se celine (moduli) smeštaju u posebne datoteke, čime se postiže lakše održavanje kompleksnih sistema.
- Moduli mogu međusobno da komuniciraju, kroz svoje interfejse.
- Modularnost — skrivanje podataka, razdvajanje poslova

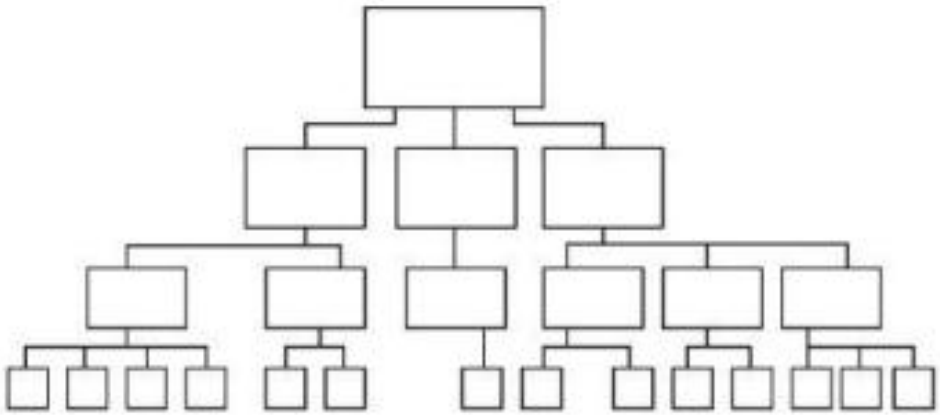
Modularna paradigma

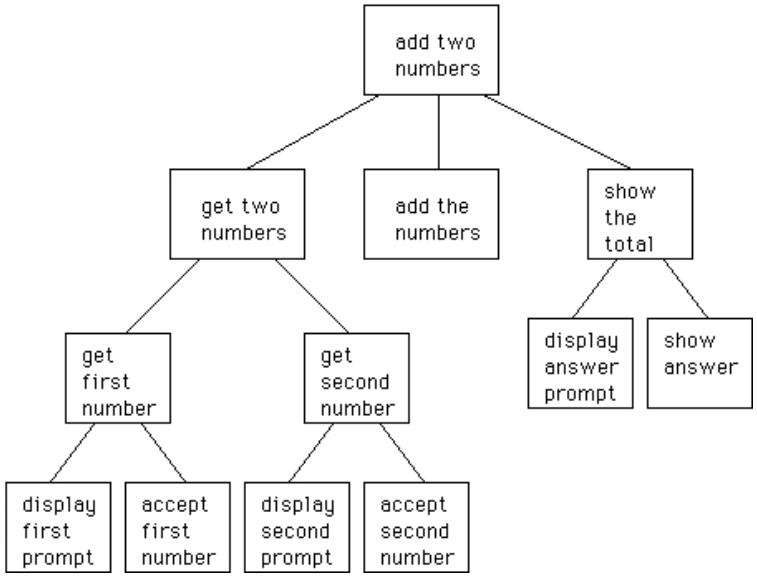
- Moduli omogućavaju višestruku upotrebu jednom napisanog koda.
- Moduli se zasebno prevode i kasnije povezuju u jedinstven program.
- Moduli u imperativnim jezicima često se nazivaju biblioteke.
- Modularnost je sada prisutna u većini programskih jezika.

Kreiranje programa od „Od opšteg ka posebnom”

- Kreiranje programa proceduralne paradigme zasniva se na principu „Od opšteg ka posebnom” (odozgo-nadole, od vrha ka dnu (Top-down concept))
- Polazi se od postavljenog zadataka kao opšteg
- Zatim se uočavaju jednostavnije celine i zadatak dekomponuje na jednostavnije delove (koji se izražavaju procedurama i funkcijama).
- Ukoliko su ti delovi i dalje kompleksni razbijaju se na još jednostavnije delove (takođe pomoću procedura i funkcija) dok se ne dodje do nivoa naredbi.
- Funkcionalna dekompozicija problema

Kreiranje programa od „Od opšteg ka posebnom”





Primer

- Napisati program koji izračunava prosečno rastojanje između zadatih tačaka u ravni.
- Opšte rešenje: učitati tačke iz datoteke, izračunati prosečno rastojanje, odštampati prosečno rastojanje.
- Sledeći korak: profiniti svaki od prethodnih koraka, posebno izračunavanje prosečnog rastojanja.
- Pronalaženje prosečnog rastojanja: sabirati rastojanja između svake dve tačke, podeliti rezultat sa ukupnim brojem parova tačaka.
- Obezbediti funkciju rastojanje
- ...

Propratni (bočni) efekti

- Prilikom izračunavanja vrednosti izraza, kod imepративnih programa česti su propratni (bočni) efekti.
- Propratni (bočni) efekti odnose se na situacije kada se prilikom izračunavanja nekog izraza istovremeno menja i stanje memorije (na primer, vrednost izraza se upisuje u neku memorijsku lokaciju)
- Oznake promenljivih su istovremeno i oznake memorijskih lokacija pa se u naredbama mešaju oznake lokacija i njihovih vrednosti, u zavisnosti od operatora koji se nad promenljivom primeni.

Propratni (bočni) efekti

- U naredbi $x = x + 1$; sa desne strane, koristi se vrednost sa lokacije x, dok se sa leve strane koristi sama lokacija na koju se upisuje sračunata vrednost.
- Propratni (bočni) efekti mogu da budu prisutni u mnogim naredbama (npr. u C-u naredbe ++, –, +=, -=, *=, ...).
- Propratni (bočni) efekti mogu značajno da otežaju razumevanje imperativnih programa.

Propratni (bočni) efekti

- Propratni (bočni) efekti odnose se i na izmenu globalnog stanja memorije nakon izvršavanja neke funkcije (u tom slučaju se kaže da funkcija ima propratne (bočne) efekte).
- Ova vrsta propratnih (bočnih) efekata je posebno nezgodna za razumevanje rada programa
- Ukoliko funkcija *sqrt* nema propratnih efekata,

  ```
  z = f(sqrt(2), sqrt(2));
  ```

  onda prethodni kod možemo transformisati na sledeći način:

  ```
  s = sqrt(2);  z = f(s, s);
  ```

  Međutim, za sledeći poziv funkcije *f*

  ```
  z = f(getchar(), getchar());
  ```

  ne možemo da ponovimo isti postupak!

Propratni (bočni) efekti

- Ukoliko funkcija ima propratne (bočne) efekte, onda to može da dovede do raznih nelogičnosti
- Na primer, izrazi:
  `2*fun()`
  `fun()+fun()`
  algebarski imaju iste vrednosti, ali u programskom jeziku ne moraju.

Primeri bočnih efekata

```
#include <stdio.h>

int a = 3;
int fun(int);
int main() {
    int f, g;
    f= a+fun(1)+a;
    g= a+fun(1)+a;
    printf("%d %d\n ", f, g);
}

int fun(int b) {
       a+=5;
       return a+b;
}
```

```
Program p;
 var z:integer;
  Function f(Var x:integer):boolean;
  begin
     x:= x+1;
     f := x > 0
  end;
  begin
    z := -1;
    writeln(f(z), '       ', f(z))
  end.
```

Pregled

Pitanja

- Pod kakvim uticajem je nastala imperativna paradigma?
- Šta je stanje programa?
- Koje su faze razvoja imperativne paradigme?
- Koje su karakteristike operacione paradigme?

Pitanja

- Koji je minimalni skup naredbi operaciona pardigme?
- Koje su karakteristike strukturne paradigme?
- Koji je minimalni skup naredbi strukturne pardigme?
- Koje su karakteristike proceduralne paradigme?

Pitanja

- Koje vrste prenosa parametara postoje?
- Kako je u memoriji orgranizovano izvršavanje potprograma?
- Šta su korutine?
- Koje su karakteristike modularne paradigme?

Pitanja

- Šta omogućava modularna paradigma?
- Kako se rešavaju problemi u okviru proceduralne paradigme?
- Šta su bočni efekti?
- Do čega dovode bočni efekti?

Literatura

- Programming Language Pragmatics, Third Edition, 2009 by Michael L. Scott
- Deo materijala je preuzet od prof Dušana Tošića, iz istoimenog kursa.