PISMENI ISPIT:

* **Odgovori na zadatke bez rješenja s FERweb ispita:**

ZAVRŠNI ISPIT 2013:

3. (ovo šta je već implementirano nije napravljeno da radi rekurzivno po cijeloj listi, nego je napravljeno tako da se može istestirati jel radi ovo šta se treba napraviti)

public static List<string> DirectoriesList = new List<string>();

string[] GetFiles(string absoluteDirectory)

{

//throw new NotImplementedException(); //već implementirano prema zadatku

return Directory.GetFiles(absoluteDirectory);

}

public static string[] GetSubdirectories(string absoluteDirectory)

{

//throw new NotImplementedException(); //već implementirano prema zadatku

return Directory.GetDirectories(absoluteDirectory);

}

static List<string> FindFiles(string absoluteDirectory, string pattern)

{

string[] directories = Directory.GetDirectories(absoluteDirectory);

var ListOfFiles = new List<string>();

Parallel.ForEach(directories, r => DirectoriesList.Add(r));

Parallel.ForEach(directories, r =>

{

try

{

string[] datoteke = Directory.GetFiles(r);

ListOfFiles.AddRange(datoteke.Where(dat => dat.Contains(pattern)));

}

catch (UnauthorizedAccessException)

{

return;

}

});

return ListOfFiles;

}

private static void Main(string[] args)

{

var list = FindFiles("F:/", "IMAG");

foreach (string el in list)

{

Console.WriteLine(el);

}

Console.ReadKey();

}

4. a) (**ispravljen, falaaa kolegi na pomoći, program je testiran u Visual Studiju**)

let zadanaLista = [1; 2; 3; 4; 5; 6]

let isPrime n =

match n with

| \_ when n > 3 && (n % 2 = 0 || n % 3 = 0) -> false

| \_ ->

let maxDiv = int(System.Math.Sqrt(float n)) + 1

let rec f d i =

if d > maxDiv then

true

else

if n % d = 0 then

false

else

f (d + i) (6 - i)

f 5 2

let rec remove\_if l predicate =

match l with

| [] -> []

| x::rest -> if predicate(x) then

(remove\_if rest predicate)

else

x::(remove\_if rest predicate)

let novaLista = remove\_if zadanaLista (fun x -> (isPrime x))

let rezultat = List.rev(novaLista)

LJETNI ROK 2013:

4.

static class Program

{

public static bool MyContains<T>(this ICollection<T> kolekcija, T element)

{

return kolekcija.Contains(element);

}

public static ICollection<T> GetDistinctValues<T>(this ICollection<T> lista)

{

ICollection<T> novaLista = new List<T>();

foreach (var el in lista)

{

if(!novaLista.Contains(el)) novaLista.Add(el);

}

return novaLista;

}

static void Main(string[] args)

{

ICollection<int> lista = new List<int>() { 1, 1, 4, 3, 3, 10 };

lista = GetDistinctValues<int>(lista);

foreach(var el in lista) Console.WriteLine(el);

Console.WriteLine();

Console.WriteLine("MyContains - " + lista.MyContains<int>(130));

Console.ReadKey();

}

}

6.

let nums1 = [5;1;2;9;11;30;20;51]

let nums2 = [8;1;3;19;121;340;250;451]

let nums1Sort = List.sort(nums1)

let nums2Sort = List.sort(nums2)

let sortFunction list1 list2 =

List.sort (List.concat[list1; list2])

let sortBrojevi = sortFunction nums1Sort nums2Sort

8. **(zasad riješeno bez korištenja broja zadataka)**

public static Dictionary<double, double> Koordinate = new Dictionary<double, double>();

public static int TockeUnutarKruga = 0;

static bool TockaUKrugu(double x, double y)

{

if (x\*x + y\*y < 1)

{

return true;

}

return false;

}

static double IzracunajPI(int brojTocaka, int brojZadataka)

{

Random varij = new Random();

for (int i = 0; i < brojZadataka; i++)

{

Thread thread = new Thread(delegate()

{

//kod

});

thread.Start();

}

for (int i = 0; i < brojTocaka; i++)

{

double f1 = varij.NextDouble();

double f2 = varij.NextDouble();

Koordinate.Add(f1, f2);

}

Parallel.ForEach(Koordinate, r =>

{

if (TockaUKrugu(r.Key, r.Value)) TockeUnutarKruga++;

else return;

});

return (4.0 \* TockeUnutarKruga/brojTocaka);

}

static void Main(string[] args)

{

Console.WriteLine("Unesi broj tocaka: ");

int broj = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Unesi broj zadataka: ");

int brojZadataka = Convert.ToInt32(Console.ReadLine());

double rez = IzracunajPI(broj, brojZadataka);

Console.WriteLine(TockeUnutarKruga + " " + rez);

Console.ReadKey();

}

* **Aspektno orijentirano programiranje**

1. *OnMethodBoundaryAspect* - Usmjeren na metode, omogućava dodavanje funkcionalnosti prije i poslije izvođenja, te uslijed bacanja iznimke i po uspješnom izvođenju, Ponašanje se definira nadjačavanjem metoda OnEnter, OnExit, OnException i OnSuccess
2. *OnExceptionAspect* - Učahuruje hvatanje iznimki u metodama i razredima za koje je nadležan, Obrada iznimki se vrši u metodi OnException
3. *OnFieldAccessAspect* - Presreće sve operacije pisanja i čitanja nad određenom varijablom, Nadjačavanjem metoda OnGetValue i OnSetValue možemo proširiti ili potpuno izmijeniti čitanje/pisanje varijable
4. *OnMethodInvocationAspect* - Kao i OnMethodBoundary, omogućava dodavanje dodatnog ponašanja metodama, Ne modificira originalnu metodu nego presreće poziv metode i pokreće druge programske linije umjesto originalnih, Željena funkcionalnost se implementira nadjačavanjem metode OnInvocation
5. *CompositionAspect* - Omogućava ubacivanje sučelja u postojeći tip, Sučelje se ne implementira nego se koristi implementacija iz nekog drugog objekta koji ga već implementira
6. *ImplementMethodAspect* - Omogućava ubacivanje implementacije apstraktnih i vanjskih (eng. external) metoda

Atribut [Serializable] je nužno navesti pri definiciji svih aspekata

ZI 2015. 2. zadatak ( ako se dobro sjećam, bio je da nađemo broj samoglasnika u riječi )

let counter list =

let rec countVowels listn sum =

match listn with

| head::tail -> match head with

| 'a' | 'e' | 'i' | 'o' | 'u' -> let nSum = sum + 1

countVowels tail nSum

| \_ -> countVowels tail sum

| [] -> sum

countVowels list 0

let rijec = "aeiuoblblblae"

let uListu = List.ofSeq(rijec)

let samoglasnika = counter uListu

Varijanta ako su nas tražili da izbacimo samoglasnike:

let isNotVowel char =

match char with

| 'a' | 'e' | 'i' | 'o' | 'u' | 'A' | 'E' | 'I' | 'O' | 'U'-> false

| \_ -> true

let rijec = "aeiuoblblblae"

let uListu = List.ofSeq(rijec)

let bezSamoglasnikaLista = List.filter isNotVowel uListu

let bezSamoglasnika = System.String.Concat(Array.ofList(bezSamoglasnikaLista))

**Alternativno rješenje s micanjem samoglasnika (antetmc2, radi u VS):**

let rec remove\_if l predicate =

match l with

| [] -> []

| x::rest -> if predicate(x) then

(remove\_if rest predicate)

else

x::(remove\_if rest predicate)

let implode (xs:char list) =

let sb = System.Text.StringBuilder(xs.Length)

xs |> List.iter (sb.Append >> ignore)

sb.ToString()

let ulaz = "varijabla VARIJABLA"

let lista = [for c in ulaz -> c]

let listaa = remove\_if lista (fun x -> ((x) = 'a' || (x) = 'e' || (x) = 'i' || (x) = 'o' || (x) = 'u'

|| (x) = 'A' || (x) = 'E' || (x) = 'I' || (x) = 'O' || (x) = 'U'))

let maknutiSamoglasnici = implode listaa

USMENI ISPIT:

* *3 načina spremanja podataka na stog (polja, načini pohrane na stog)*

Dinamički oblik (pohrana na hrpi uz pokazivač na stogu), statički oblik (pohranu polja u aktivacijski zapis bloka), oblik određen u trenu deklaracije (korištenjem opisnika - dope vektora) **(nisam siguran)**

* *A-liste ( Asocijacijske liste, ista stvar )*

Veze objekta i imena se pohranjuju u asocijacijskim listama (zvanim i A-listama) koje se implementiraju stogom. **DINAMIČKI DOSEG** ( slajdovi 39-40, Teorijski 05, slika to lakše objašnjava )

Osim izravno u aktivacijskim zapisima, spomenute veze objekata i imena mogu se spremati i u A-liste koja se održava kao stog. (dodatak - knjiga **Programming Languages, Principles and Paradigms**)

Laički: Lista pokazivača u parovima, jedan pokazivač u paru pokazuje na objekt, drugi na idući pokazivač u paru. Gradi se od vrha prema dnu. Na slici se sve dosta jasno vidi.

* *Aspektno programiranje*

Aspektno orijentirano programiranje (eng. Aspect-oriented programming) je programska paradigma koja povećava modularnost dodatnim razdvajanjem poslova (eng. concerns) unutar programa. Aspektno programiranje dopunjuje OO programiranje uvođenjem dinamičkih elemenata u statičke modele. Nove/dodatne funkcionalnosti se nalaze na jednoj lokaciji i ne kvare OO model.

* *Closure, problemi kod closurea (nepostojanje okoline nakon vracanja funkcije)*

Prilikom poziva po imenu u funkciju se prenosi tekstualna reprezentacija izraza i okolina u kojoj se on izvršava U trenu kada pozvana procedura treba izvršiti izraz, dohvaća se mehanizam zatvaranje (eng. closure) i izvršava se u okolini iz koje je funkcija pozvana.

U slučaju apstraktnog stroja sa stogom, mehanizam zatvaranje se sastoji od: Pokazivača na kod poslan formalnim parametrom , Pokazivača na statički lanac pokazivača koji vodi do okoline u kojoj se evaluira izraz iz formalnog parametra .Mehanizam zatvaranja se uz formalni parametar veže pokazivačem ( slajd 14, Teorijski 07 ).

Laički: Kolko sam ja skužio, ovo je vrlo slično delegatu. Praktički se pozvana funkcija prebaci u pozivajuću funkciju ( tu je u igri taj closure ) uz zamjenu imena varijabli. Ispravite me ako griješim.

* *Copy kod garbage collectora*

Kod tzv. stani-kopiraj sakupljača (eng. stop and copy collector) hrpa se dijeli u dva područja jednake veličine. Pri normalnoj uporabi je u jednom trenu aktivno samo jedno područje što znači da je jedno područje aktivno, a jedno područje neaktivno. Zauzimanja memorije se rade u samo jednom (aktivnom) području – Kada se memorija u aktivnom području iscrpi, pokreće se sakupljač smeća koji prolazi izvorno memorijsko područje i kopira memorijske objekte u odredišno (drugom, neaktivno) memorijsko područje zaredom eliminirajući fragmentaciju. Kad se pojedini objekt kopira, sve reference sadržane u tom objektu su ažurirane da pokazuju na nove lokacije referenciranih objekata. Također, pošto se u neaktivni kopiraju samo živi (korišteni) objekti, u aktivnom dijelu ostaje smeće, tj. ostaju objekti koji se ne koriste. – Po završetku kopiranja, aktivno područje postaje neaktivno i obrnuto, u neaktivnom se području oslobađa memorija od neaktivnih objekata (smeća), a program nastavlja s izvršavanjem ( slajd 41, Teorijsku 08 )

* *CRT tablica*

Svi blokovi programa pokazuju na središnju tablicu (CRT tablicu) u koju se pohranjuju sva imena korištena u programu Uz svako se ime pohranjuje i zastavica s informacijom o aktivnosti imena te pokazivač na lokaciju s ostalim informacijama o imenu (tip, memorijska lokacija itd.) Imena koje se koriste u programu mogu (korištenje fiksnog odmaka od početka tablica) i ne moraju biti poznata u trenu prevođenja programa (korištenja raspršenog adresiranja) Ulaz i izlaz iz blokova se ostvaruje zasebnim stogovima vezanima za svaku varijablu pojedinačno. **DINAMIČKI DOSEG** ( slajdovi 41-42, Teorijski 05, slika to lakše objašnjava )

* *Currying*

Curryjev postupak (currying) je mogućnost parcijalnog izvršavanja funkcije s više parametara.

npr. Definiramo funkciju s 2 parametra. Tu funkciju je moguće klasično pozvati s 2 parametra, ali je također moguće stvoriti i novu funkciju tako da se originalna funkcija pozove sa samo jednim parametrom. **FUNKCIJSKO PROGRAMIRANJE**

* *Dinamičke liste (aktivacijski stog) ( možda se misli na dinamički stog/aktivacijski zapisi )*

Pokazivač svakog aktivacijskog zapisa (eng. activation record pointer) pokazuje na točno određeno područje prethodnog zapisa Aktivacijski zapisi se stavljaju i skidaju sa stoga u trenu izvođenja programa Upravljanje stogom u trenu izvođenja programa se izvodi odsječcima programa koji se umeću prije i nakon poziva procedure i početka i kraja bloka koji se poziva.

( slajdovi 26-27, Teorijski 05 )

* *Dinamičke liste kod dosega*
* *Dinamički lanci pokazivača i koja im je veza s aktivacijskim postupkom kod poziva funkcije*

Prilikom poziva procedure ili ulaska u programski blok se na stogu alocira memorijski prostor koji se zove **aktivacijski zapis ili okvir**. On se sastoji od međurezultata izračuna, lokalnih varijabli i lanaca dinamičkih pokazivača, lanaca statičkih pokazivača, povratnih adresa, povratnih rezultata i parametara. Lanci dinamičkih pokazivača se sastoje od pokazivača na prethodni aktivacijski zapis na stogu. Oni ulančavaju aktivacijske blokove po pozivu.

* *Display kod statičnog*

Rješavanje problema implementacije statičkog dosega statičkim lancem kod kojeg je potrebno napraviti određen skokova po lancu u svrhu razlučivanja ne-lokalnih imena. Tehnika prikaza (eng. display) koristi vektor u kojem element na poziciji k sadrži pokazivač na aktivacijski zapis na nivou gniježđenja k koje je trenutno aktivno Kada se referencira ne-lokalni objekt koji je deklariran izvan bloka k, pristupa se aktivacijskom zapisu na kojeg pokazuje kta pozicija vektora ( slajdovi 37-38, Teorijski 05, slika to lakše objašnjava )

* *DOPE vektor i single free lista*

**DOPE vektor (opisnik)-** struktura podataka unutar modela za alokaciju polja u memoriji koja služi za čuvanje informacija o nekom objektu

**Jednostruka slobodna lista** (single free list) -> u velikom memorijskom prostoru hrpe zauzimaju se varijabilni memorijski prostori redom kako dolaze zahtjevi za zauzimanjem. Postoje dva načina korištenja dealocirane memorije:

* *Direktno korištenje slobodne liste* - Lista se pretražuje kako bi se pronašao prostor jednake ili veće veličine od traženog. Dva su načina odabira slobodnih blokova: prvo podudaranje i najbolje podudaranje.
* *Slobodno sažimanje memorije* - svi aktivni blokovi se miču na kraj nastojeći sve slobodne blokove objediniti u jedan.
* *dynamic dispatch u kontekstu polimorfizma*

Dynamic dispatch je proces prilikom kojeg se prilikom izvođenja programa odabire koja implementacija polimorfne funkcije/metode će se pozvati. Static dispatch odabire prilikom prevođenja programa.

* *Funkcijski jezici, funkcijsko programiranje*

Funkcijsko programiranje je [programska paradigma](https://hr.wikipedia.org/w/index.php?title=Programska_paradigma&action=edit&redlink=1) koja tretira [računanje](https://hr.wikipedia.org/wiki/Ra%C4%8Dunanje) kao evaluaciju [matematičkih funkcija](https://hr.wikipedia.org/wiki/Funkcija_(matematika)) i izbjegava stanje i [mutabilne](https://hr.wikipedia.org/w/index.php?title=Nemutabilni_objekt&action=edit&redlink=1) podatke. Naglašava primjenu funkcija, kao suprotnost stilu [imperativnog programiranja](https://hr.wikipedia.org/w/index.php?title=Imperativno_programiranja&action=edit&redlink=1) koji naglašava promjene u [stanju](https://hr.wikipedia.org/w/index.php?title=Stanje_programa&action=edit&redlink=1). ( Wikipedija )

Funkcijski jezici: F#, Haskell

**Karakteristike funkcijskog programiranja**: lijena evaluacija (vrsta evaluacije karakteristična za funkcijske programske jezike), podatkovna apstrakcija, podudaranje uzorka, zaključivanje o tipovima (nije potrebno specificirati posebne tipove konstrukata), svođenje na izračunavanje funkcija, nepromjenjivost programskih struktura te im se vrijednost može dodijeliti samo prilikom stvaranja, Curryjev postupak.

* *Garbage kolektori (copy i reference) copy gore objašnjen*

Zajedno s objektom na hrpi rezervira se i brojač referenci (eng. reference counter) koji nije direktno dostupan programeru Apstraktni stroj mora osigurati da se u brojaču referenci nalazi broj koji označava koliko je pokazivača na taj objekt još aktivno Po stvaranju objekta, brojač se postavlja na 1 Po izlasku iz lokalne okoline, brojači svih objekata na koje pokazuju pokazivači iz lokalne okoline se smanjuju za 1 U trenu kad neki brojač postane 0, memorija pripadnog objekta se može osloboditi Taj objekt može sadržavati i interne pokazivače pa apstraktni stroj mora obići i njihove interne pokazivače i rekurzivno obaviti smanjivanja svih povezanih brojača ( slajd 38, Teorijski 08, slika dodatno objašnjava ).

* *Koja je alternativa stogu za dinamičko povezivanje* - **CRT**
* *Lambda izrazi ( lambda funkcije? )*

Funkcije koje se definiraju i pozivaju bez eksplicitnog navođenja identifikatora Zovu se i anonimne funkcije.

let add = fun a b -> a + b;;

add 2 3 ( rezultat 5 )

* *Lanac statičkih/dinamičkih pokazivača*

Redoslijed pregledavanja aktivacijskih zapisa kako bi se pronašlo ne-lokalno ime NE ovisi o njihovoj poziciji na stogu, već ovisi o (tekstualnoj) strukturi programa Aktivacijski su zapisi povezani lancem statičkih pokazivača (eng. static chain pointer) koji se koristi za implementaciju statičkog dosega Tipično, uz lanac statičkih pokazivača, prisutan je i lanac dinamičkih pokazivača (eng. dynamic chain pointer) u cilju ulančavanja aktivacijskih blokova po pozivu ( slajdovi 33-34, Teorijski 05, slika to lakše objašnjava )

Laički: Statički pokazivači pokazuju strukturu cijelog programa ( koja funkcija/metoda je definirana unutar koje funkcije/metode ) prilikom prevođenja, dok dinamički pokazivači pokazuju strukturu cijelog programa prilikom izvođenja.

* *lijena evaluacija*

Lijena evaluacija znači da se izrazi evaluiraju samo onda kad i ako je to potrebno. Uvijek se evaluiraju samo operandi koji su potrebni te se to odlučuje prilikom predaje neevaluiranih operanada operatoru.

* *Regularni izrazi (Pattern Matching), s kojim se znakom povezuje više regularnih izraza*

**Preklapanje uzoraka** (pattern matching) je uspoređivanje odabranog ulaznog konstrukta sa skupom uzoraka istog tipa. Najčešće se ostvaruju putem ključnih riječi match i with te je njime omogućeno složeno grananje. Više regularnih izraza se povezuje znakom | nakon čega slijedi zapis uzorka (jednog od po volji mnogo).

* *Statička polja*
* *Statičke liste u C-u*
* *strukturna ekvivalentost*

**Pravilo referencijalne prozirnosti**: kod strukturne se ekvivalencije dva ekvivalentna tipa u bilo kojem kontekstu mogu međusobno zamijeniti bez da se promijeni smisao programa u kojem se zamjena obavlja. Programski jezici kombiniraju ekvivalenciju po imenu i strukturnu ekvivalenciju.

* *Šta je side-effect funkcija*

Side-effect funkcije su problem popratnih pojava koji se javlja kada funkcija ili izraz uz vraćanje vrijednosti modificira stanje ili ima vidljivu interakciju s funkcijama koje poziva ili s vanjskim svijetom. Uvijek utječu na varijable koje su izvan izraza koji se izračunava. Većinom se javlja u funkcijskim jezicima.

Npr. funkcija modificira statičku ili globalnu varijablu

* *Što je aspekt i od čega se sastoji*

Aspekti su zasebni moduli koje odvaja aspektno orijentirano programiranje, a služe za rješavanje problema omogućavanjem izdvajanja problematičnih poslova u zasebne module. Aspekt se sastoji od kombinacije savjeta ( Dodatne programske linije koje želimo pridružiti postojećem modelu ) i prijelomnih točaka ( Točka u izvođenju programa kojoj želimo pridodati dodatne funkcionalnosti u svrhu rješavanja problema ). ( slajd 8, Praktični dio 08 )

* *Što su unije u C-u i po čemu se razlikuju od struktura?*

Unije u C-u su analogne strukturama u definiciji i izboru polja s razlikom u tome što samo jedno od polja može biti aktivno u određenom vremenu s obzirom da polja (koja mogu biti različitog tipa) koriste isti prostor pohrane (isto memorijsko područje). (knjiga **Programming Languages, Principles and Paradigms**) Često i diskriminanta promjenjivog zapisa određuje varijante zapisa koje promjenjiva unija može poprimiti.

* *Univerzalni parametarski polimorfizam i nadovezanje na to*

Neka vrijednost ima univerzalno parametarsko višeobličje kada postoji konačan broj različitih tipova koje se može koristiti uporabom jedne jedine scheme općenitog tipa Jedan programski odsječak koji uniformno radi nad svim instancama općenitog tipa (informacija konkretnog tipa se ne koristi u algoritmu) Objekt s univerzalnim parametarskim višeobličjem se može instancirati za neki specifični tip Vrste: Eksplicitno parametersko višeobličje (npr. predlošci (eng. templates) u C++ ili generici (eng. generics) u Javi, Implicitno parametarsko višeobličje (npr. u ML) ( slajd 29, Teorijski 08 )

Laički: kolko sam ja skužio, to bi bila funkcija il nekakav programski odsječak koji radi nad svim donekle sličnim tipovima varijabli ( tipa funkcija Zamjeni(<T> a, <T> b) koja bi za širok spektar tipova varijabli mogla zamijeniti vrijednost te dvije varijable ). Također se takav objekt može instancirati za specifični tip ako nam treba.

* *Unutarnja i vanjska fragmentacija*

**Unutarnja fragmentacija** (eng. internal fragmentation) je pojava kod koje se rezervira memorijski blok koji je veći od memorije koju je program zatražio

**Vanjska fragmentacija** (eng. external fragmentation) se javlja kada su svi slobodni memorijski blokovi manje veličine od veličine bloka koji se želi rezervirati, a suma ukupno dostupne memorije je veća ili jednaka traženoj. **HRPA** ( slajd 30 )

* *Viseće reference i kako ih riješiti (nadgrobne ploče; ključ i ključanica)*

Problem visećih referenci se javlja kad dvije ili više varijabli/pokazivača pokazuju na istu vrijednost/zapis u memoriji, a nakon toga jednu od varijablu dealociramo i oslobodimo memorijski prostor. Prilikom pristupa toj nepostojećoj vrijednosti kod ostalih varijabli dobit ćemo grešku.

**Nadgrobne ploče:** Korištenje dodatnog memorijskog prostora koji se zove nadgrobna ploča Zauzima se svaki puta kada se uzima adresa memorijske lokacije na stogu Prilikom pristupa adresi, apstraktni stroj ubacuje novu razinu indirekcije kojom se preko nadgrobne ploče pristupa traženom podatku Prilikom oslobađanja objekta, nadgrobna ploča se deaktivira Nadgrobne ploče se nalaze u specifičnom dijelu memorije koje se naziva groblje (eng. cemetery) ( slajd 35, Teorijski 08, slika vrlo dobro objašnjava ).

**Ključ/ključanica:** Prilikom stvaranja svakog objekta na hrpi, isti se povezuje s ključanicom (eng. lock) u koji se pohranjuje neka nasumična vrijednost Uz pokazivač se pohranjuje ista vrijednost koja predstavlja ključ (eng. key) Prilikom dohvata neke vrijednosti preko pokazivača (dereferenciranja) , uspoređuju se vrijednost pohranjena uz ključ i uz ključanicu. – Ako su vrijednosti jednake, nastavlja se rad – U suprotnom, ne dozvoljava se pristup memorijskoj lokaciji Prilikom pridruživanja jednog pokazivača drugome, prepisuje se cijeli par (adresa i ključ) Prilikom dealokacije memorije, vrijednost u ključanici se obezvrjeđuje ( slajd 36, Teorijski 08, slika vrlo dobro objašnjava ).

**NEKA PITANJA SA STARIH USMENIH (nespomenuta ili s drugačijim odgovorima):**

* *Jensenov uređaj*

Jensenov uređaj predstavlja poziv funkcije s prijenosom izraza i varijable u paru po imenu pri čemu je varijabla dio izraza te promjene varijable mijenjaju i vrijednost izraza.

* *Dinamičko upravljanje memorijom korištenjem hrpe*

Ono se koristi kod jezika koji imaju eksplicitne naredbe za alokaciju memorije. Koristi se hrpa koja je memorijski prostor u kojem se memorijski blokovi mogu relativno slobodno alocirati i dealocirati. Podijeljena je u više elemenata (blokova) relativno male veličine povezanih u listu koju nazivamo **slobodnom listom**.

U dobrom dijelu slučajeva okolina izvođenja programa dozvoljava rezervaciju memorijskih blokova varijabilne veličine na hrpi pri čemu se koriste mehanizmi koji ubrzavaju rezervaciju memorije i brzinu izvršavanja operacija upravljanja memorijom na hrpi. Želi se izbjeći pojava fragmentacije -> unutarnja i vanjska fragmentacija (*već opisani gore*).

Tehnike alokacije memorije uključuju mehanizme spajanja slobodnih memorijskih blokova radi eliminacije vanjske fragmentacije.

Mehanizmi spajanja memorijskih blokova su: jednostruka slobodna lista (*već opisana gore*) i višestruke slobodne liste - korištenje različitih lista za blokove različitih veličina: sustav prijatelja i Fibonaccijeva hrpa.

* *Razlika između proceduralnih i funkcijskih jezika*

Kod proceduralnih se jezika definira kako će nešto biti izvršeno, dok se kod funkcijskih jezika definira što će biti izvršeno. **(nisam siguran)**

* *S čim imaju veze veličine blokova kod dinamičkog upravljanja memorijom*

Imaju veze s tehnikama alokacije memorije i fragmentacijama.

* *Varijabilni tipovi*
* *Shallow & Deep Binding i kako se ostvaruje Deep Binding*

Povezivanja se javljaju prilikom pozivanje funkcije f višeg reda koja je predana preko formalnog parametra h. Postoje dvije mogućnosti odabira ne-lokalne okoline:

* **Korištenje dubokog povezivanja (Deep Binding)** - Korištenje okoline koja je aktivna kada se stvara veza između stvarnog i formalnog parametra. Koriste ga jezici sa statičkim i dinamičkim dosegom.
* **Korištenje plitkog povezivanja (Shallow Binding)** - Korištenje okoline koja je aktivna kada dođe do poziva funkcije koja je predana formalnom parametru. Koriste ga samo jezici s dinamičkim dosegom.

Kod implementacije Deep Bindinga je najvažnije da se informacija o statičkom lancu pokazivača mora odrediti u trenutku povezivanja stvarnih i formalnih parametara. Formalni parametar koji prima funkciju se mora povezati s kodom funkcije i s nelokalnom okolinom u kojoj se tijelo funkcije evaluira. Prilikom poziva funkcije preko formalnog parametra, tijek se programa preusmjerava na prvi element mehanizma zatvaranja, drugi se koristi kao statički pokazivač aktivacijskog zapisa nove okoline.

* *Razlika između funkcijskih i imperativnih jezika*

Za funkcijske jezike je karakteristična lijena evaluacija, dok je kod imperativnih jezika karakteristična trenutna evaluacija. Također imperativni jezici koriste varijable kojima se može mijenjati vrijednost, dok kod funkcijskih jezika postoji nepromjenjivost programskih struktura. U imperativnim jezicima vrijednost funkcijskog tipa ne može biti rezultat kompleksnih izraza, dok u funkcijskim jezicima može biti. Funkcijski jezici naglašavaju primjenu funkcija, dok imperativni jezici naglašavaju promjene u [sta](https://hr.wikipedia.org/w/index.php?title=Stanje_programa&action=edit&redlink=1)n[ju](https://hr.wikipedia.org/w/index.php?title=Stanje_programa&action=edit&redlink=1).

* *Što je nasljeđivanje i zašto ga nije dobro puno upotrijebljavati*

Nasljeđivanje je odnos između klasa kod kojeg se jedna klasa stvara na temelju druge tako da joj se dodaju specifični atributi i ponašanje. Ako se koristi previše nasljeđivanja, kod postaje težak za čitanje, razumijevanje, praćenje, održavanje, debuggiranje, itd. Također se vrijeme izvođenja usporava jer se prilikom poziva pozivaju sve metode nadklasa.

* *Pohranjivanje višedimenzionalnog polja u memoriji*

Moguće je definirati multidimenzionalna polja koja se onda mogu pohranjivati kao multidimenzionalna polja i polja polja (u Pascalu su identični, dok je u C-u multidimenzionalno polje zapravo polje polja). Mehanizmi provjere tipa trebaju osigurati da se svaki pristup elementu spremljenog polja uistinu dogodi unutar granica i da se pristupa postojećoj vrijednosti.

* *Poziv po imenu (navedeni i načini poziva funkcija)*

Načini poziva funkcija su: **poziv po vrijednosti** (Prilikom poziva funkcije stvarni se parametar evaluira i njegova se vrijednost povezuje s formalnim parametrom. Pri završetku funkcije okolina funkcije i formalni parametar se uništavaju.) i **poziv po referenci** (Stvarni parametar je izraz koji se prilikom poziva funkcije evaluira te se veza stvarnog i formalnog parametra unosi u lokalnu okolinu procedure. Svaka promjena formalnog parametra je i promjena stvarnog parametra). Varijacije poziva po referenci su: **poziv po konstanti, poziv po rezultatu, poziv po vrijednosti-rezultatu, poziv po imenu**.

**Poziv po imenu** - Varijacija poziva po vrijednosti, koristi pravilo kopiranja: Neka je funkcija f funkcija s jednom parametrom x i neka je a izraz koji je tipa kompatibilnog s tipom od x. Poziv funkcije f sa stvarnim parametrom je semantički ekvivalentan izvršavanju tijela funkcije f u kojem se sva pojavljivanja formalnog parametra x zamjenjuju s a. Supstitucija izraza se provodi prilikom evaluacije u okolini pozivajućeg programa.

* *Objasniti svojstva funkcijske paradigme (već objašnjeni dijelovi su manje detaljniji)*
* **Nepromjenjivost** - Programske strukture u funkcijskim jezicima su nepromjenjive što znači da im se vrijednost može dodijeliti samo prilikom stvaranja. Stanje se nakon stvaranja nikako ne može promijeniti.
* **Zaključivanje o tipovima** - Konkretne tipove konstrukata u funkcijskim jezicima nije potrebno specifirati.
* **Rekuzija** - Nije specifična samo za funkcijsko programiranje, ali se često koristi. Također se može koristiti rekurzija repa.
* **Preklapanje uzoraka (pattern matching)** - Uspoređivanje odabranog ulaznog konstrukta sa skupom uzoraka istog tipa čime se omogućuje složeno grananje (slično kao if-else s razlikom da match-with moraju pokriti sve moguće slučajeve).
* **Lijena evaluacija** - Vrsta evaluacije karakteristična za funkcijske programske jezike gdje se računanje ne obavlja sve dok zaista nije potrebno. U F#-u je podrazumijevana vrsta evaluacije trenutna evaluacija, ali se može uključiti i lijena evaluacija.
* **Curryjev postupak** - Mogućnost parcijalnog izvršavanja funkcije s više parametara.