VARIJABLE

deklarirati varijable bliže mjestu prvog korištenja

- varijabla se neće promijeniti(slučajno) ako je blizu deklaracije
- ako deklariramo na početku dobiva se dojam da se koristi u cijelom programu
- ako premještamo blok koda obuhvatit ćemo i deklaraciju

doseg (scope) - područje programa di možemo koristiti varijablu(vidljiva nam je varijabla)
smanjenje dosega:

- varijable koje se koriste u petlji inicijalizirat odmah prije petlje
- ostale varijable inicijalizirat prije prve upotrebe
- grupirat naredbe koje koriste iste varijable
- grupe naredbi premjestit u postupke
- počet s što manjim dosegom pa ako treba tek širit

prostor ranjivosti - kod koji se nalazi između dvije upotrebe neke varijable, dakle sav kod koji ne koristi neku varijablu, a omeđen je kodom u kojem se koristi ta varijabla -> ciljamo da bude što manji
prostor uporabe (variable span):

```
a=0;
b=0;
c=0;
b=a+1;
b=b/c;
```

redovi razmaka između korištenja varijable b: 1 i 0
račun: (1+0)/2 = 0.5
vrijeme života (live time) - prostor od prvog do zadnjeg korištenja neke varijable (primjer gore od b=0; (linija 2) do b=b/c; (linija 5) dakle vrijeme života varijable b je 5 - 2 + 1 = 4
prednosti kratkog života varijable

- smanjuje se prostor ranjivosti
- čitljiviji, jasniji kod
- manja šansa da ćemo fulat kod inicijalizacije varijable
- lakše se kod može premještati

vrijeme povezivanja(binding time) - vrijeme kad se varijabla poveže s vrijednosti

- u trenu pisanja - hardcodiranje
- u trenu prevođenja - postavljanje vrijednosti u varijablu iz programa
- u trenu izvršavanja - postavljanje vrijednosti u varijablu učitavanjem

loše:

- jedna varijabla za više svrha (tmp varijabla)
- varijable sa skrivenim značenjem (varijablom se dostavlja podatak ili određena vrijednost ak je došlo do greške)
- neiskorištene varijable (deklarirane, ali nikad korištene)

imenovanje varijabli

- opisno imenovanje (ime varijable opisuje sadržaj)
- duljina - optimalno 10-16 znakova

- kratka imena (i,j,k) dozvoljena kod petlji, ali ako je petlja u petlji ili se varijabla koristi van petlje treba opisno
- logičke varijable (primjer dobrih): statusOK, found (NE KORISTITI negativna imena poput notFound - valjda da bi bili optimisti -.-)

kakva imena izbjegavati:

- kratice koje se mogu pogrešno interpretirati (višestruko značenje)
- imena sa sličnim značenjem (recordNum, numRecord)
- imena koja se razlikuju u jednom ili dva znaka (dakle veoma slična)
- imena koja slično zvuče
- imenima s brojevima (osim ak nije nužno npr. dio naziva)
- pravopisno neispravna imena
- riječi koje drugi često ne znaju pravopisno napisati
- imena kojima je jedina razlika velka/mala slova
- imena koja se podudaraju s ključnim riječima (nije moguće u fakin normalnim kompajlerima)
- imena koja sadrže znakove koji se teško razlikuju u nekom fontu (npr. I, L, 1)

OSNOVNI TIPOVI PODATAKA

magični brojevi - hardcodirani brojevi - bolje rješenje -> upotreba konstanti, ali dozvoljeno je koristiti 0 ili 1 hardcodirano

konstante

- promjene na jednom mjestu za svaku upotrebu broja u programu
- podatak je opisan prek imena
- korisno kod npr. definiranja veličine polja

pravila za rad s brojevima

- paziti na dijeljenje s nulom
- naglasiti pretvorbe tipova
- izbjegavati usporedbe drugačijih tipova

preljev (overflow) - prekoračenje granica mogućnosti pohrane tog tipa (npr. 16 bitni int ide od -32768 do 32767)

pogreška u zapisu broja u računalu - ograničena točnost

smjernice za rad sa znakovnim nizovima u C/C++:

- inicijalizirat s null characterom
- koristiti statička polja
- koristiti funkcije koje imaju ograničenja po broju znakova (npr. strncpy() umjesto strcpy())

logičke varijable(statusne):

- bolje dokumentiranje koda
- pojednostavljenje if-ova i case-ova (logičkih izraza)

pobrojani tip (enum):

- čitljivost
- pouzdanost
- lakša modifikacija
- alternativci za logičke varijable
- pokriva se mogućnost unosa nedozvoljene vrijednosti

nizovi

- svi indeksi koje koristimo moraju biti unutar granica (ne smijemo van polja)
- provjeriti granične elemente
- cross-talk - kad se indeksi pomiješaju npr. unutar višestruke petlje kad idemo po matrici pa zamijenimo red i stupac
- u pravilu bolje izbjegavati nizove :D ak ih ne trebamo (probat listu il tak neš fancy)

IZRAZI I KONTROLE NAREDBE

ogranizacija koda:

- imenovati postupke da opiše sve što postupak radi i da ukazuje na međuovisnost
- međuovisnost koda mora biti očita
- koristiti parametre kako bi međuovisnost bila očitija (staviti da postupak prima varijable i vraća umjesto npr void postupka koji radi s globalnim varijablama)
- komentirati ak niš drugo ne upali

princip bliskosti - držati povezane naredbe skup(manje vrijeme života varijabli)

smjernice za pisanje if-a:

- prvo napisati glavni slučaj, a tek onda sporedne
- provjeriti da su usporedbe dobre (npr. da nije < umjesto <=)
- glavni slučaj uz if, a ne uz else

umjesto if - else if - else sa puno slučajeva bolja case naredba (urednije)

ako switch-case ima hrpu slučajeva preporuča se jedan od poredaka:

- abecedno, numerički
- glavni slučaj prvi
- po tome kak se često koriste (glavni slučaj prvi) - preglednost, brzina

smjernice za korištenje switch-case:

- što manje naredbi u granama
- ne izmišljati nepotrebne varijable samo da bi koristio case
- default granu koristiti za vrijednosti koje nisu očekivane, ili dojavu greške ako se ne mogu obraditi te vrijednosti
- izbjegavati propadanje (koristiti break;) - ako se koristi onda to dokumentirati, al u pravilu izbjegavati

PETLJE

while

- testiranje na početku
- testiranje u sredini - paziti na razumljivost, upotrijebiti umjesto izvršavanja dijela petlje pomoću goto naredbe
- testiranje na kraju

for

- poznat broj izvršavanja
- bez kontrolne naredbe za izlaz
- ne mijenjati indekse da bi se prijevremeno svršilo (bolje onda while)

ulazak u petlju

- samo na jednom mjestu u kodu
- inicijalizacije neposredno ispred petlje
- za beskonačne petlje while(true)
- pazit da je upotreba for-a prikladna

unutar petlje

- izbjegavat prazne petlje
- uvijek stavljati {} oko tijela petlje
- jedna petlja -> jedan zadatak
- housekeeping (rad s pomoćnim varijablama) postavit na početak ili kraj petlje

izlazak iz petlje

- paziti da petlja sigurno završava
- naglasit (učinit očitim) izlaz iz petlje
- opet ono za for... ne smiju se mijenjat indeksi da bi završilo prije
- ne koristit završnu vrijednost indeks varijable nakon petlje
- koristiti break i continue umjesto logičkih varijabli
- ne koristiti puno break naredbi - razlomi u više petlji

petlje for dummies

- počni s tijelom petlje
- dodaj petlju oko tijela
- dodaj ak treba petlje oko vanjske petlje
- paziti na promijene u tijelu petlje pri dodavanju još jedne petlje oko petlje
- dodat inicijalizacije prije petlje

POSTUPCI

zakaj su postupci naši prijatelji

- smanjuju kompleksnost koda - sakrivamo neki dio koda o kojem nećemo razmišljat previše, a radi... lakše održavanje, pregledno, lakše shvatit
- uvode apstrakciju - samodokumentiranje koda, koristeći jednu naredbu(poziv) koristimo sve naredbe unutar postupka
- izbjegavamo dupli kod - ako imamo sličan kod u više postupaka trebamo ih učinit istima i spojit u jedan
- jednostavnije testiranje logičkih testova - dokumentiranost, lakše izmjene, preglednost
- centralizacija kontrole
- reusability
- poboljšanje performansi

čak i kratki postupci su ok, jer razvijanjem programa će se vjerojatno povećat

kohezija - jedan postupak = jedan zadatak

vrste kohezije

- funkcionalna - postupak radi točno jedan zadatak
- sekvencijalna - postupak sadrži zadatke koji se moraju točnim redoslijedom izvršavati -> treba izdvojiti zadatke u posebne postupke
- komunikacijska - zadatke povezuju isti ulazni podaci -> izdvoji u posebne postupke
- vremenska - zadaci se izvršavaju u isto vrijeme -> izdvojiti u posebne postupke i imenovati da se naglasi vremensku koheziju
- proceduralna - zadaci se bez razloga izvršavaju u određenom slijedu -> izdvoji u posebne postupke koje će jedan postupak povezivati
- logička - nekoliko zadataka se ugura u jedan postupak, a izvršavaju se ovisno o parametru -> izdvoji i pozivaj samostalno
- slučajna - zadaci nemaju niš zajedničko

imenovanje postupka

- glagol - radnja koju vršimo
- ime objekta nad kojim vršimo radnju

parametri

1. ulazni
2. koje modificiramo
3. izlazni

ako više postupaka koristi slične parametre, poredati na isti način

smjernice za korištenje parametara

- koristiti sve parametre
- statusne varijable na kraj
- dokumentirati što se pretpostavlja o parametrima (koje vrijednosti, jesu li ulazni ili izlazni)
- ne više od 7 parametara

- parametre ne koristiti kao varijable u računu unutar postupka
- koristiti imenovane parametre

KAKO PISATI KOD

smjernice za formatiranje koda

- format mora naglasiti logičku strukturu
- konzistentno
- čitljivo
- jednostavne modifikacije

blokovi

- čisti - basic stil (nema zagrada nego keywords)
- emulacija čistih blokova (početna zagrada u liniji s keywordom)
- s eksplicitnim granicama (početna zagrada liniju nakon keyworda)
- blokovi na kraju naredbi - nije čitljivo (poput basica ali završni keyword je u ravnini s krajem početne naredbe)

jedna naredba po retku

- čitljivost
- više naredbi u jednom retku ne daje optimiziraniji kod
- lakše naći grešku
- lakše debuggirat
- lakše mijenjat kod

UVOD U OOP

osnovni koncepti:

- apstrakcija - modeliramo objekt koristeći samo bitna svojstva tog objekta
- učahurivanje (enkapsulacija) - skrivanje atributa i ponašanja od drugih klasa - postiže se neovisnost objekata
- nasljeđivanje - klasa se stvori na temelju druge klase dodavanjem specifičnih atributa i ponašanja
- višeobličje (polimorfizam) - isto ponašanje drugačije definirano za različite klase
- asocijacija (povezanost)
- agregacija - sastavljeno od više drugih objekata

UVOD U .NET FRAMEWORK

.NET program (npr. C#)

↓

MSIL - stogovni međujezik

↓

CLR - common language runtime

↓

strojni kod

CLR

- provjerava ima li dozvole za izvršavanje u trenutnom kontekstu programa
- učitava program (sprema u cache)
- oslobađa resurse - garbage collector, sprječava memory leak
- provodi JIT prevođenje
    1. program (npr. C#) prevodi u MSIL
    2. prilikom izvršavanja MSIL prevodi u strojni kod
    3. svako sljedeće izvršavanje uzima strojni kod iz cachea
- višejezičko okruženje (CTS - common type system, zajednički tipovi podataka za .NET jezike, nasljeđivanje kroz više jezika)
- hvatanje iznimki (try-catch)
- asembliji - najmanja cjelina izvršnog koda, izvršna datoteka ili biblioteka
- jednostavna instalacija
    - jednostavno kopiranje asemblija na novu lokaciju
    - CAB za automatsku instalaciju
    - instalacija pomoću windows instalera
- podrška verzioniranju kroz asemblije

BCL - base class library

- razredi grupirani u namespace (npr. System.Data, System.Collections)
- .NET sučelje prema windows API-u

CLS - common language specifications

UVOD U C#

boxing - pakiranje vrijednosnih (varijabla) tipova u referentne(objekt)

namespace (prostori imena) - pretinci u koje se grupiraju tipovi, istoimeni moraju biti u različitim namespaceovima

object ima:

- ToString - pretvara objekt u string
- Equals - uspoređuje 2 objekta
- GetHashCode - vraća jedinstvenu identifikaciju objekta
- GetType - vraća tip objekta
- MemberwiseClone - stvara novu instancu objekta i kopira vrijednosti atributa

kopiranje objekata

- duboka kopija - kopira sve elemente
- plitka kopija - kopira samo referencu

argumenti metoda

- referentni tipovi - prenose se kao reference
- vrijednosni tipovi - vrijednosti ili reference (ref - ulazno izlazni argument, out - izlazni)

argumenti

- bez prefiksa - promjene nisu vidljive van metode
- out - metoda ga smije postavljati, ali ne čitati
- ref - metoda ga smije i čitati i postavljati

višeznačnost (overloading) - više metoda istog imena s različitim parametrima (poziv se bira na temelju parametara)

OSNOVE OOP-A U C#

modifikatori vidljivosti

- private - nije vidljivo van klase
- public - vidljivo
- protected

javno sučelje - javne metode - komunikacija s drugim objektima/metodama

konstruktor - poziva se automatski prilikom stvaranja objekta klase (ako nismo napisali konstruktor uzima se defaultni - bez parametara)

List<TypeOfElement> - klasa koja predstavlja listu

- Add(TypeOfElement elementName)
- Clear()
- Sort()

statička varijabla - zajednička za sve objekte(jedna referenca za sve), nije potrebno instancirati klasu