XII - Standardne biblioteke u C-u

- Veliku pomoć prilikom programiranja u C jeziku pružaju nam prateće datoteke koje uključujemo u program kao **datoteke zaglavlja**
- Svaka **datoteka zaglavlja** sadrži:
  1. jednu ili više deklaracija funkcije,
  2. funkcijske prototipove bibliotečnih funkcija,
  3. deklaracije tipova podataka,
  4. makro naredbe,
  5. simboličke konstante,
  6. globalne promenljive koje se koriste sa bibliotečnim funkcijama
- Datoteke zaglavlja se uključuju u program pomoću naredbe:

  #include <zaglavlje>
- Ova naredba se nalazi izvan bilo koje deklaracije ili definicije i pre korištenja onoga što deklarira.
- Sledeći slajdovi daju popis svih zaglavlja i pripadajućih funkcija standardne biblioteke koji su definisani u ANSI C jeziku.

XII-Standardne biblioteke u ANSII C

assert.h – definiše assert makro za dijagnostiku programa

ctype.h – sadrži informaciju koja se koristi za klasifikaciju znakova i makroe za pretvaranje znakova (isalnum,.. iscntrl,.. ispunct,.. toupper…)

errno.h – sadrži funkcije za testiranje i podešavanje indikatora statusa ili pogrešaka i definicije simbola za kodove pogrešaka (clearerr, ferror, perror, feof)

float.h – sadrži konstante za funkcije sa pokretnim zarezom. Svaka implementacija definiše vrednost tih konstanti.

limits.h – sadrži parametre okruženja i raspone veličina integralnih tipova.

locale.h – deklariše funkcije koje osiguravaju informacije zavisne o zemlji i jeziku.

math.h – deklariše matematičke funkcije i makroe (sin, cos, .. floor, .. ldexp….).

setjmp.h – definiše tip jmp_buf i funkcije longjmp i setjmp koje koriste taj tip (setjmp, longjmp).

XII-Standardne biblioteke u ANSII C

signal.h – defiše konstante i deklaracije koje se koriste u signal i raise f-jama (signal, raise).

stdarg.h – definira makroe koji se koriste za čitanje liste argumenata u funkcijama deklariranim da prihvate promjenjivi broj argumenata (va_list, va_start, va_end)

stddef.h – definira nekoliko općih tipova podataka i makroa

stdio.h – definira tipove, makroe i std. U/I tokove stdin, stdout, stderr i deklarira funkcije U/I toka (fopen,... fprintf, .. ftell, .. fsetpos)

stdlib.h – deklarira nekoliko općih funkcija: za pretvorbu tipova, pretraživanje, sortiranje (atof, .. malloc, .. abort, .. labs, ldiv….)

string.h – deklarira nekoliko funkcija za manipuliranje stringovima i memorijom (strcpy, .. , strstr, .. , memchr, .., memset….)

time.h – definira strukturu koja se popunjava f-jama pretvorbe vremena i tip koji se koristi u funkcijama ctime, difftime, gmtime, localtime i stime, te sadrži njihove prototipove (clock, mktime, gmtime, time, asctime, localtime, difftime, ctime, strtime).

XII – Važnije funkciju u biblioteci math.h

Ime funkcije		Ime funkcije	Opis
acos	Inverzni kosinus	log	Prirodni logaritam
acosh	Inverzni hiperbolički kosinus	log2	Logaritam po bazi 2
asin	Inverzni sinus	log10	Lograitam po bazi 10
asinh	Inverzni hiperbolički sinus	pow(x,y)	xy
atan	Inverzni tangens	round	Zaokruživanje realnog broja
atanh	Inverzni hiperbolički tangens	sin	Sinus
cbrt	Kubni koren	sinh	Hiperbolički sinus
cos	Kosinus	sqrt	Kvadratni koren
cosh	Hiperbolički kosinus	tan	Tangens
exp	Eksponencijalna funkcija	tanh	Hiperbolički tangens
exp2(x)	2x		
fabs	Apsolutna vrednost realnog broja		
fmax(x,y)	Vraća veću vrednost od x i y		
fmin(x,y)	Vraća manju vrednost od x i y		
hypot(x,y)	Hipotenuza - sqrt(x2 + y2)		

XII - Otkrivanje grešaka-Debugg

- U programima koje pišemo obično mogu da se jave dva tipa greške:
  1. **sintaksne** (*compile-time*)
  2. **izvršne** (*run-time*) greške.
- **Sintaksna** ispravnost programa podrazumeva da je program zapisan u skladu sa specifikacijom programskog jezika, tj. da su sve naredbe programa ispravne jezičke konstrukcije u odnosu na gramatiku jezika.
- Čak i kada je program sintaksno ispravan, to ne znači da je on ispravan
- Program može da sadrži greške logičke prirode - **semantičke greške**.
- Otkrivanje i ispravljanje **semantičkih(izvršnih) grešaka** je daleko teže od otkrivanja i ispravljanja **sintaksnih grešaka**.
- Popularni engleski naziv za ove greške u programu je ***bug*** (buba)
- Mogu se koristiti dve tehnike za pronalaženje izvršnih grešaka:
  1. **štampajuće instrukcije** - ubaciti izlazne instrukcije na ključnim tačkama programa , tj. korišćenje - **WRITE tehnika**
  2. program ***debugger***, omogućava kontrolisani rad programa

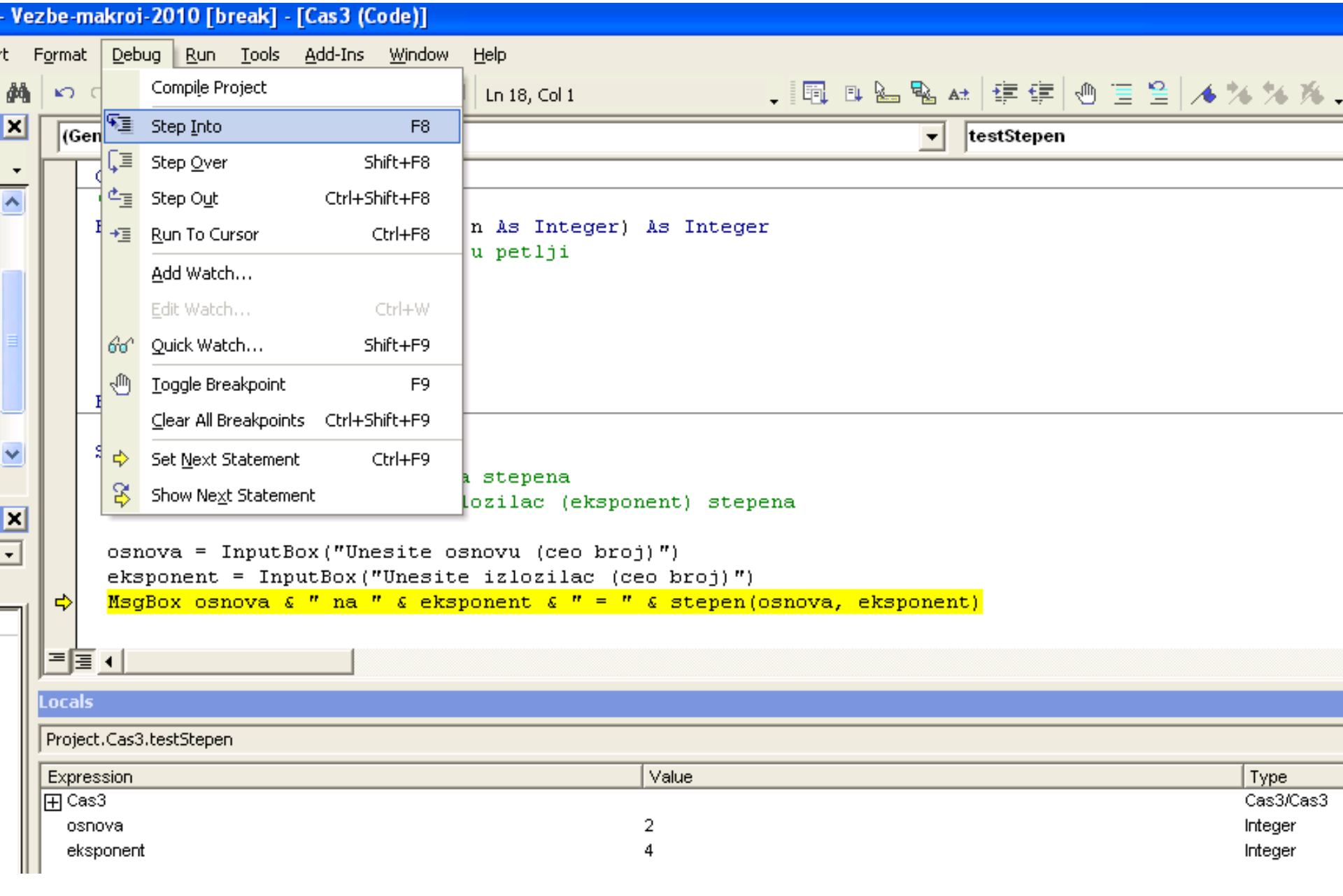
XII - Otkrivanje grešaka-Debugg

- Značaj otkrivanja i ispravljanja semantičkih grešaka je vremenom doveo do razvoja posebnih softverskih alata čiji je to zadatak.
- Da bi se olakšao proces razvoja softvera, danas za razne programske jezike postoje **integrisana razvojna okruženja** (*Integrated Development Environment*, IDE) koja u sebi objedinjuju alate za:
  1. unos i uređivanje naredbi programa pomoću tastature - **Text editor**
  2. prevođenje/intepretiranje programa
  3. otkrivanje i ispravljanje sintaksnih grešaka
  4. otkrivanje i ispravljanje semantičkih grešaka – **Debugger**
- Dijagnostika i korigovanje greške, ***debugging***, je proces identifikacije uzroka tj. korena neke greške, i samo korigovanje te greške.
- Kod nekih projekata, *debugging* (korigovanje grešaka, debagiranje) nosi više od 50% od ukupnog vremena za razvoj softvera.
- Za mnoge programere *korigovanje grešaka* je **najteži deo posla**.
- Međutim, debagiranje se može učiniti mnogo lakšim ako se koriste određene tehnike koje u mnogome pomažu pronalaženju grešaka.

XII - Otkrivanje grešaka-Debugg

- Značaj otkrivanja i ispravljanja semantičkih grešaka je vremenom doveo do razvoja posebnih softverskih alata čiji je to zadatak.
- Da bi se olakšao proces razvoja softvera, danas za razne programske jezike postoje integrisana razvojna okruženja (*Integrated Development Environment*, IDE) koja u sebi objedinjuju alate za:
  1. unos i uređivanje naredbi programa pomoću tastature - **Text editor**
  2. prevođenje/intepretiranje programa
  3. otkrivanje i ispravljanje sintaksnih grešaka
  4. otkrivanje i ispravljanje semantičkih grešaka - **Debugger**
- Dijagnostika i korigovanje greške, ***debugging***, je proces identifikacije uzroka tj. korena neke greške, i samo korigovanje te greške.
- Kod nekih projekata, *debugging* (korigovanje grešaka, debagiranje) nosi 50% od ukupnog vremena razvoja softvera.
- Za mnoge programere *korigovanje grešaka* je **najteži deo posla**.
- Međutim, debagiranje se može načiniti mnogo lakšim ako se koriste određene tehnike koje u mnogome pomažu pronalaženju grešaka.

XII - Otkrivanje grešaka-Debugg



XII - Otkrivanje grešaka-Debugg

- Softverski defekti (*bugs*) su greške u softveru odnosno greške koje prave programeri (*errors*, *defects*, *faults*).
- Debagiranje omogućuje dijagnozu tih defekata i njihovo otkrivanje.
- Kod korigovanja pronađenih grešaka, može se često doći do pravljenja novih grešaka koje mogu da budu još gore/teže.
- Jedan od osnovnih načina korišćenja debugger-a je **Korak po korak**
- Predstavlja postepeno izvršavanje jedne po jedne naredbe programa (meni ***Debug > Step Into*** (F8))
- Omogućuje praćenje promene vrednosti svih promenljivih tekućeg potprograma (tj. procedure ili funkcije čija se naredba izvršava).
- Imena i vrednosti promenljivih tekućeg potprograma su dostupne u posebnom prozoru **Locals Window** ("prozor sa lokalnim promenljivama"), koji postaje vidljiv aktiviranjem istoimene opcije iz menija View.
- Međutim, nisu svi programeri vešti u **programiranju/debagiranju**.
- Najbolji dijagnostičari mogu biti tri puta brži od onih najlošijih.

XII - Otkrivanje grešaka-Debugg

- Jedno poređenje između jednog najboljeg i jednog najlošijeg programera, za slučaj gde je bilo 12 grešaka u napisanom programu.

	Najbolji programer	Najlošiji programer
Vreme dedefektovanja u minutima	5	15
Broj nepronadjenih defkata	0	4 od 12
Dodati novi defekti	0	11

- Ovo govori da je **dijagnoza greške** disciplina kojoj treba **posvetiti veliku pažnju**, pa se zato programeri dodatno obučavaju u ovoj oblasti
- Nije dovoljno reći da se dijagnoza greški može obaviti samo pomoću ubacivanja serije instrukcija štampanja u program.
- Iako je otklanjanje grešaka teško za programere, ono je veoma korisno i predstavlja i pogodnost tj. priliku da se nauči nešto korisno, naime:
  1. da se nauči nešto o tom programu,
  2. da se nauči o tipovima i uzrocima grešaka koje se prave,
  3. da se nauče tehnike dijagnoze greški,
  4. da se nauče tehnike korigovanja greški, gde se traži tačna dijagnoza i otklanjanje uzroka a ne simptoma problema

XII - Otkrivanje grešaka-Debugg

- Mnogi programeri neozbiljno prilaze pronalaženju grešaka
- U neefikasne prilaze kod dijagnoze greške kod nekog programa spada:
  1. **Pogađanje -** pomoću slučajno tj. haotično izabranih instrukcija štampanja u programu,
  2. pronalaženje greške pomoću **slučajnih/haotičnih** izmena u programu
  3. izbegavanje da se udubi duboko u problem, već se **površno pokušava ispraviti problem**
- Dijagnoza i korigovanje geške sastoji se od **nalaženja greške** ali i **popravke greške**.
- Međutim, nalaženje greške i razumevanje uzroka nastajanja greške je obično 90% posla u procesu otklanjanja same greške.
- Obično programeri koji su pisali neki softver teže dolaze do pronalaženja izvršnih (***run time***) greški u sopstvenom programu
- U mnogim softverskim kompanijama postoje posebni timovi tkz. **testeri programa** kojima je isključivi zadatak da vrše testiranje softvera koji je neko drugi napisao

XII – Krajnje mere za dijagnozu greške

- Ponekad je potrebno primeniti **krajnje mere**, koje će svakako rešiti problem dijagnoze greške ako se nije uspelo pomoću testova i hipoteza o mogućoj grešci:

1. uraditi kompletnu proveru dizajna i kodiranja dela programa koji ne radi dobro
2. odbaciti deo programa koji ne radi dobro, i ponovo ga iz početka dizajnirati i napisati (kodirati)
3. odbaciti ceo program i ponovo ga dizajnirati i kodirati iz početka
4. uraditi kompilaciju sa kompletnom informacijom o debagiranju
5. pooštriti testiranje softverskih jedinica, i njih testirati u izolaciji
6. koristiti debugger da se provere široki opsezi test parametara
7. promeniti kompajler
8. uneti u program štampanje i prikaz kritičnih promenljiva.
9. kompilaciju i izvršavanje programa obaviti negde drugde, u drugom okruženju
10. integrisati program u malim delovima, i kompletno testirati svaki taj deo programa kao samostalan program.

XII – Korigovanje grešaka

- Teže je pronaći grešku nego korigovati grešku.
- Kod korigovanja greške često se prave nove greške.
- Sledeće su **sugestije** za izbegavanje pravljenja novih greški kod korigovanja postojećih greški:

1. pre korigovanja potrebno je razumeti problem, toliko razumeti npr. da se može predvideti svako pojavljivanje te greške
2. razumeti ceo program, tj. kontekst problema, jer onda se može rešiti problem kompletno a ne delimično
3. potvrditi da je dijagnoza korektna, pre nego se pristupi korigovanju greške, tj. proveriti ispravnost hipoteze greške, odnosno odbaciti ostale hipoteze koje nisu tačne
4. memorisati originalni izvorni kod, u cilju poređenja sa novim kodom
5. korigovati problem, a ne samo simptom.
6. menjati program samo ako ste sigurni da je ispravna promena, jer ako se on menja po sistemu probe i greške, to nije uopšte efikasan metod
7. vršiti jednu po jednu promenu programa, a ne više njih istovremeno,
8. proveriti izvršene korekcije

XII – Korigovanje grešaka

- Za male programe, WRITE tehnika daje dobre rezultate.
- Kod velikih programa sa mnogo linija koda metoda štampanja ne daje dobre rezultate ali je tu za vas neki od alata za dijagnostiku
- Kod velikih programa programer ne zna gde da postavi izlazne (štampajuće) instrukcije.
- Proces dodavanja štampajućih instrukcija, pa kompilacija, pa egzekucija je vrlo dosadan tj. spor i neefikasan proces.
- Poseban problem je kasnije uklanjanje dodatih instrukcija štampanja
- Kod problema sa pokazivačima (pointers), pronalaženje greške je posebno teško jer pokazivač prikazan u HEX formatu ne znači ništa.
- Umesto WRITE tehnike može se primeniti specijalni program koji se zove **debugger**, ali je pre toga potrebno naučiti kako koristi sve mogućnosti ovog programa.
- Na sledećem slajdu date su komande koje se mogu koristiti u debugger
- Ove komande su raspoložive u okviru Visual C++, Netbeans IDE i Code::Blocks okruženja:

XII – Tipične komande Debugerr-a

Start executing in debugger	startovanje programa pomoću *debugger*-a
Step in:	izvršavanje jedne instrukcije tj. *single-stepping* (koračanje)
Step over (Next step):	preskakanje metode koračanja
Continue:	nastaviti izvršavanje programa
View variable:	prikazivanje promenljive
Set breakpoint:	zaustavljanje programa u određenoj instrukciji
Add watch:	prikazati promenljivu kad se program zaustavi
Program reset:	startovanje programa iz početka pomoću *debugger*-a

XII- Prednosti i mane alata za dijagnostiku

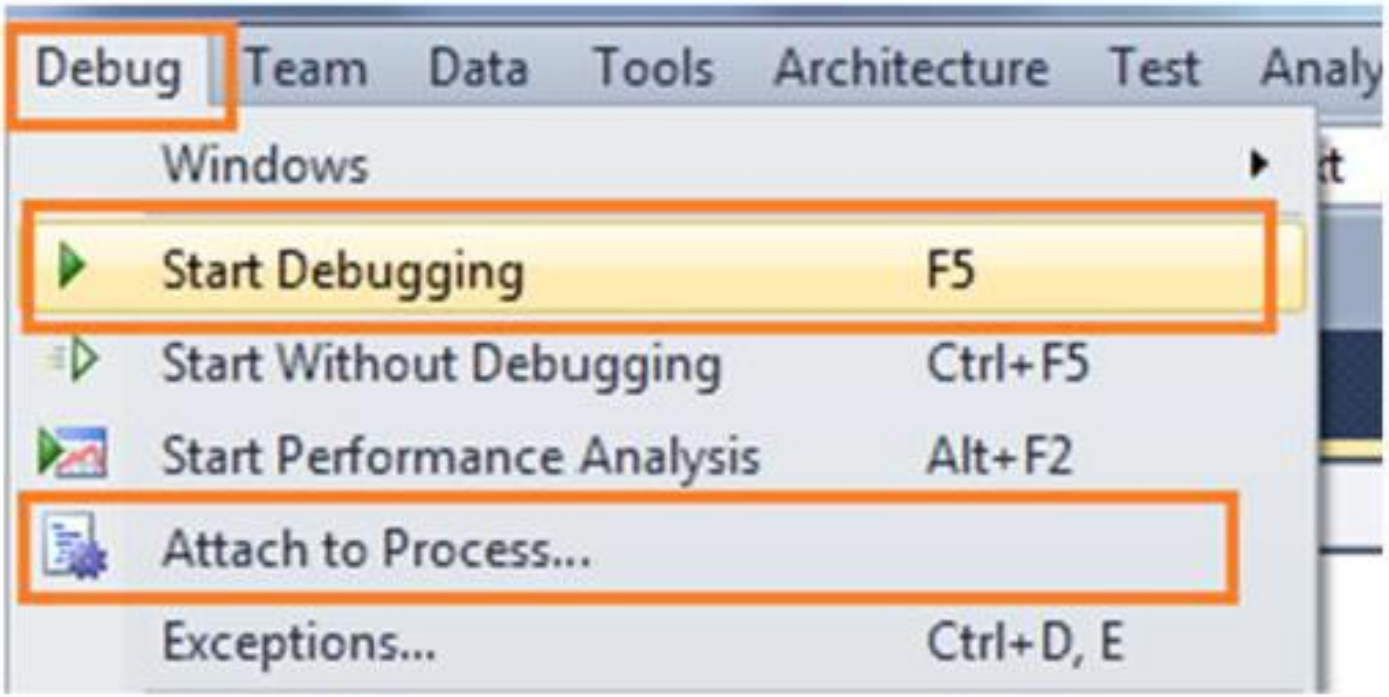
- Alati za dijagnostiku su korisno sredstvo ali pod uslovom da programer ima dobru kontrolu nad kodom koji testira i da je svestan tipa problema koji je nastao
- Programi za dijagnostiku greške, **debugger**-i, omogućuju:
  1. da se postave zaustavne tačke u programu, da se program jednostavno zaustavi kod određene linije u programu ili da se zaustavi kada se nekoj promenljivoj zadaje vrednost
  2. omogućuju izvršavanje programa liniju po liniju, tzv. koračanje kroz program ili podprogram, ili preskakanje podprograma
  3. omogućuju izvršavanje programa unazad do tačke gde je defekt započeo.
  4. omogućuju štampanje raznih iskaza u toku izvršavanja programa, npr ‘pogledaj ovde‘ ili slično
  5. pomoću njih se mogu ispitivati podaci u programu , npr. dinamički alocirana matrica podataka.
- Postoje kontroverzna mišljenja o dijagnostičkim programima.

XII- Prednosti i mane alata za dijagnostiku

- Neki programeri su protiv njihove upotrebe, argumentujući da se defekti brže i tačnije otkrivaju pomoću razmišljanja i ranije opisanih tehnika, i da se mentalnom egzekucijom programa, a ne dijagnostičkim programom, trebaju otkrivati defekti.
- Ovo mišljenje da treba odbaciti dijagnostičke programe **nije u redu**.
- Ali isto tako, treba reći da se svako oruđe može koristiti ispod ili iznad svojih mogućnosti, i to važi i za dijagnostičke programe.
- Dakle, ne možemo se potpuno osloniti na dijagnostičke programe misleći da će oni umesto vas otkriti greške.
- Dijagnostički program je samo jedno pomoćno sredstvo, i on nije zamena za razmišljanje kao glavni alat dijagnostike greške.
- Ali ni razmišljanje nije totalna zamena za dijagnostički program.
- Kod interaktivne dijagnostike se koristi **debugger** da se menjaju razni parametri u programu u toku izvršavanja programa.
- Međutim, interaktivni debugger ima nedostatak a to je da ohrabruje prilaz **pokušaj-i-pogreši**, ***trial-and-error***, tj haotično otkrivanje greški umesto sistematskog prilaza.

XII - Rad sa Debugerom

- Debagiranje programa može započeti tako što se iz menija Debug izabere "***Start Debugging***" ili jednostavno pritiskom na F5
- Ukoliko ste u nekoj od linija naveli tačku zaustavljanja (***breakpoint***) izvršenje programa će početi i program će se pauzirati u toj liniji.
- U suprotnom će se program izvršiti do kraja ili do linije gde će se desiti neočekivani prekid.



XII - Postavljanje Breakpoint

- **Tačka zaustavljanja – Breakpoint** se koristi sa ciljem da se Alat za korigovanje obavesti kada i gde treba da pauzira program u toku rada
- Tačka zaustavljanja se postavlja tako što postavimo kursor na liniji u kojoj želimo da se program zaustavi i pritisnemo taster F9
- U trenutku kad debugger stigne u tačku zaustavljanja programer dobija mogućnost da ispita šta nije u redu sa kodom korišćenjem alata za ispravljanje grešaka.

main.c

(Global Scope)

```
{
        printf("Unesite ime osobe:");
        gets(o->ime);
        printf("Unesite adresu osobe:");
        gets(o->adresa);
        printf("Unesite razred i odeljenje ucenika:");
        scanf("%u%u",&o->raz,&o->ode);
        while(getchar()!='\n');
}
```

XII - Rad Korak po korak

- ***Step over*** i ***Step Into*** omogućavaju praćenje programa korak po korak.
- ***Step Over*** odmah odrađuje liniju poziva funkcije

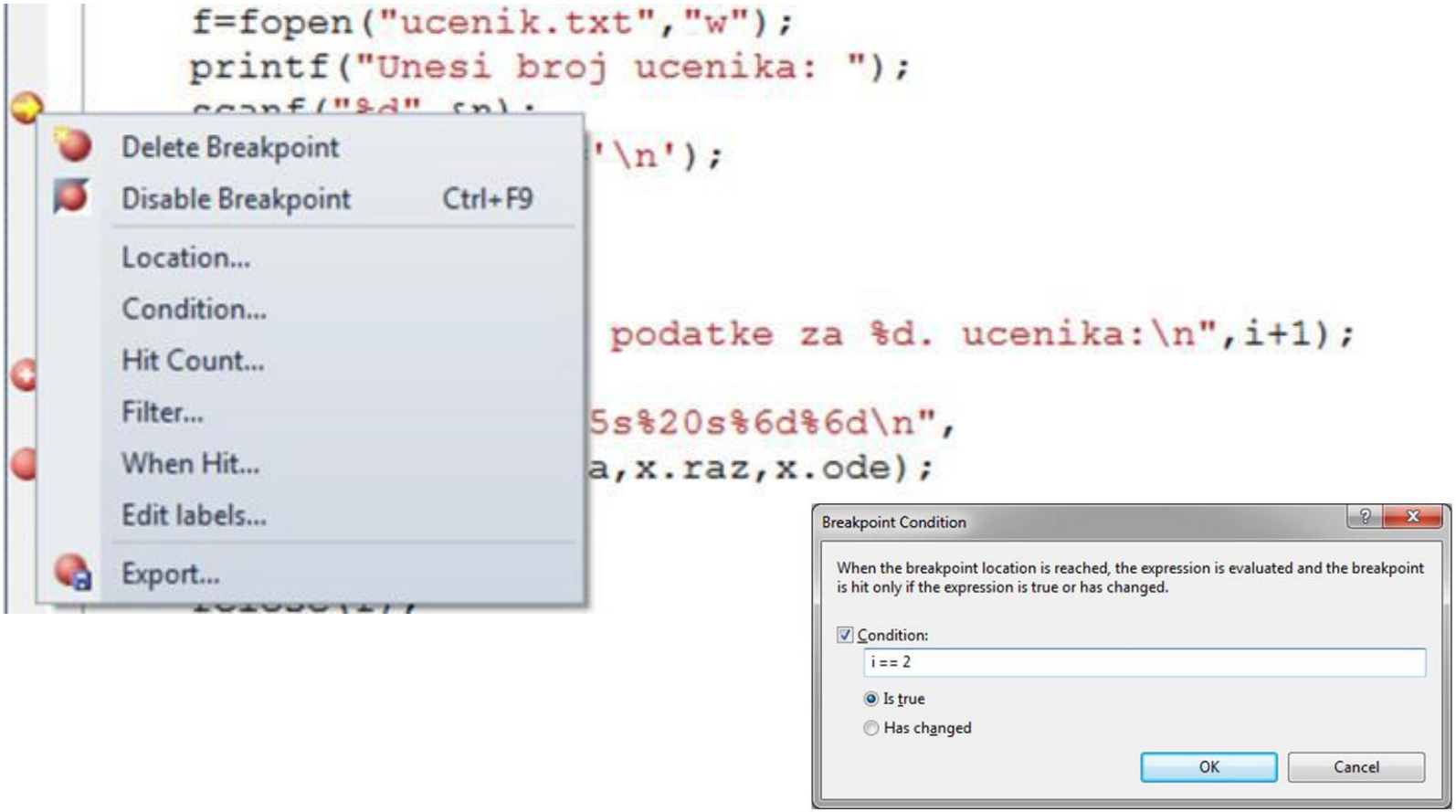
```
for(i=0;i<n;i++)
{
    printf("Unesi podatke za %d. ucenika:\n",i+1);
    citaj(&x);
    fprintf(f,"%15s%20s%6d%6d\n",
    x.ime,x.adresa,x.raz,x.ode);
}
```

- dok ***Step Into*** ulazi u prvu liniju pozvane funkcije

```
void citaj(struct ucenik *o)
{
    printf("Unesite ime osobe:");
    gets(o->ime);
    printf("Unesite adresu osobe:");
    gets(o->adresa);
    printf("Unesite razred i odeljenje ucenika:");
    scanf("%u%u",&o->raz,&o->ode);
    while(getchar()!='\n');
}
```

XII – Uslovna tačka prekida

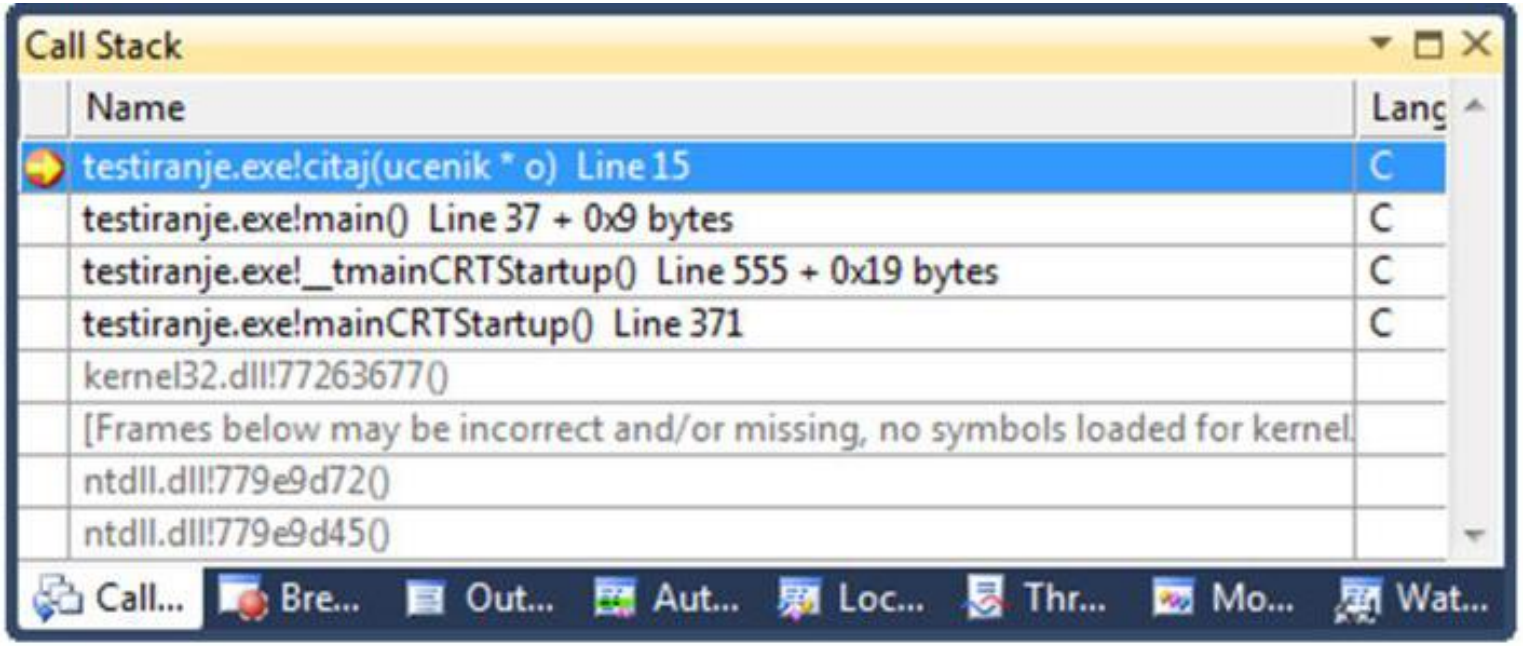
- **Uslovna tačka prekida** zaustavlja rad ako je zadovoljen neki uslov.
- Da bi smo ovo uradili neophodno je da prvo postavimo tačku prekida
- Zatim pritisnemo desni taster miša na crvenu tačku koja označava tačku prekida (breakpoint) i otvoriće se dopunski meni
- U okviru njega treba zadati uslov (***Condition***) prekida rada programa



XII – Prozori za praćenje

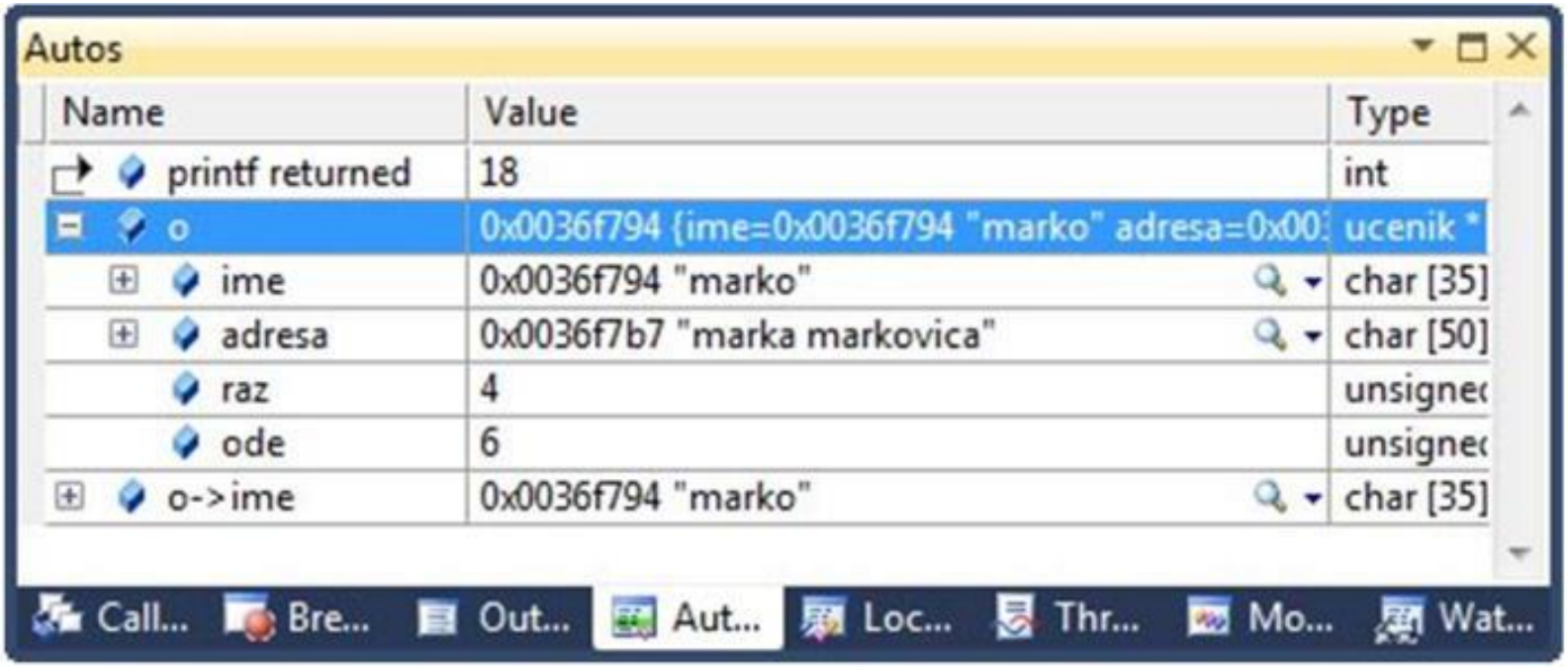
- Prozori za praćenje omogućavaju praćenje promenljivih u trenutku kada debugger dostigne tačku zaustavljanja tj. **breakpoint**
- Omogućavaju da postavljanjem kursora miša preko imena promenljive dobijate dodatne informacije o toj promenljivoj tip podatka, vrednost..

- Prozor **Call Stack**



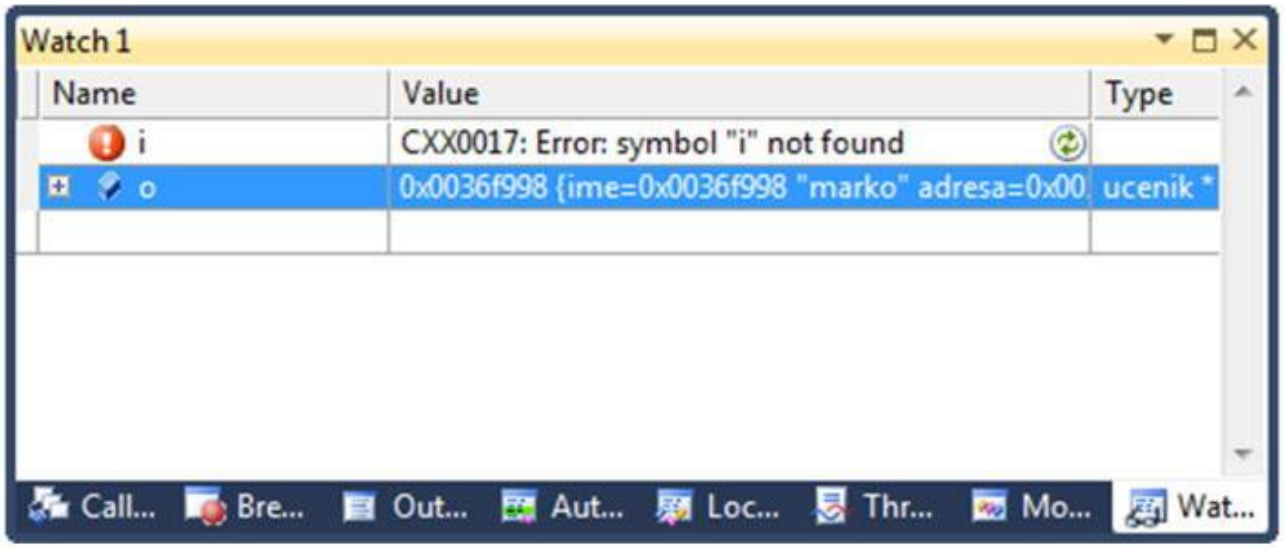
XII – Prozori za praćenje

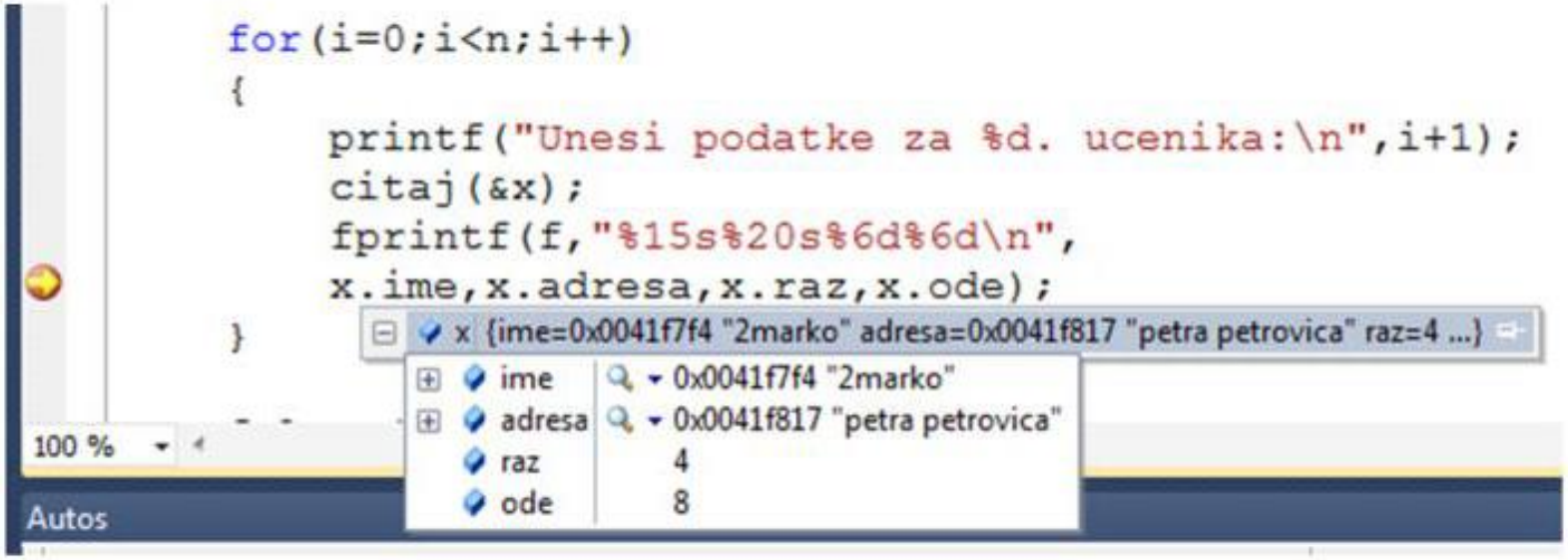
- Prozor **Autos**



XII – Prozori za praćenje

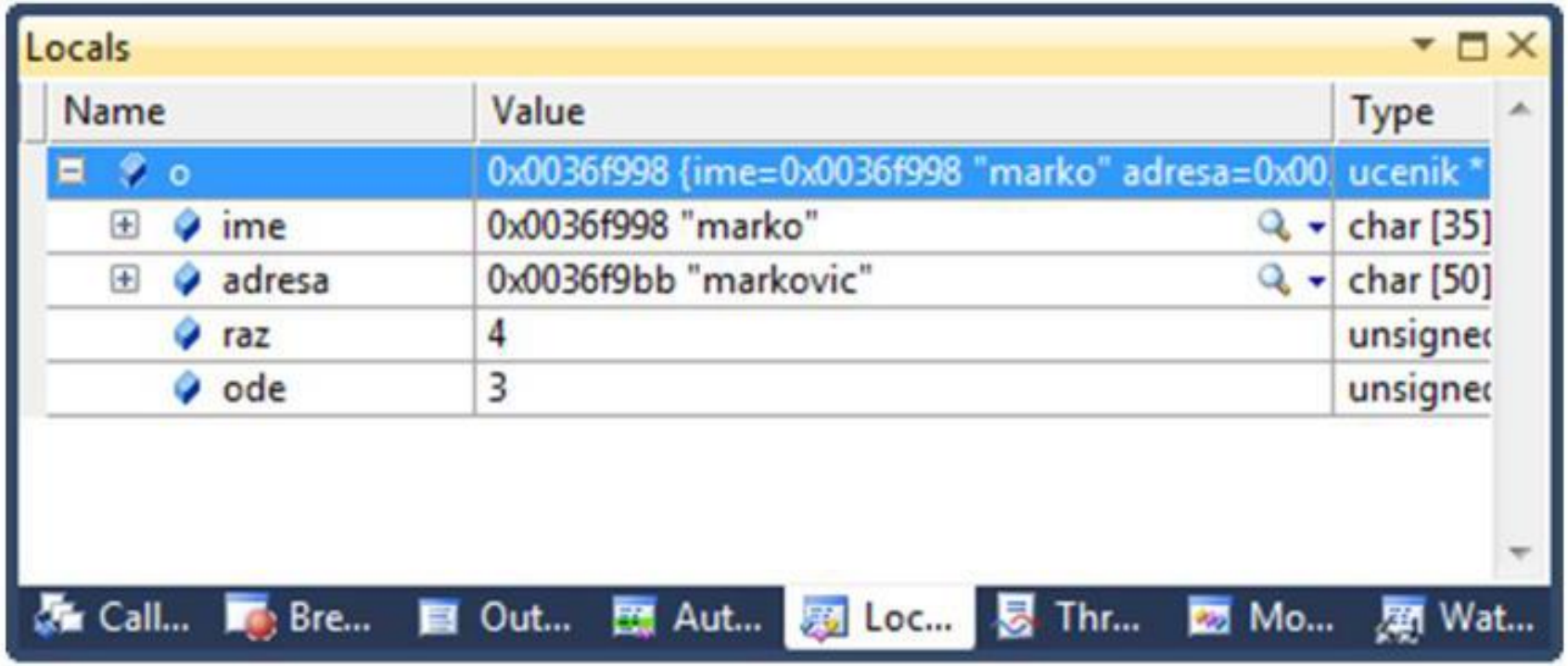
- Prozor **Watch** se koristi za praćenje pojedinačnih objekata ili promenljivih





XII – Prozori za praćenje

- U prozoru **Locals** prate se lokalne promenljive



Hvala na pažnji !!!

Pitanja

? ? ?