Odbrambeno programiranje

Razvoj softvera 2

Kosta Đurišić

Sadržaj

- Šta je odbrambeno programiranje?
- Zaštita programa od lošeg unosa
- Assert
- Tehnike obrade grešaka
- Izuzeci
- Ograđivanje štete u programu izazvane greškama u kodu
- Debugging Aids
- Koliko odbrambenog programiranja ostaviti u završnom kodu

Šta je odbrambeno programiranje?

- Prepoznavanje da će vaš program imati probleme i modifikacije, i pisanje koda sa tom idejom na umu.
- Analogija: Bezbedna vožnja

Zaštita programa od lošeg unosa

- ″Garbage in, garbage out″ nije dovoljno dobro.
- U dobrom programu:
  - 1.″Garbage in, nothing out″
  - 2.″Garbage in, error message out″
  - 3. ″No garbage allowed in″

Zaštita programa od lošeg unosa

- Postoje 3 uopštena načina kako da obrađujemo loše ulazne podatke :
  - 1. Provera vrednosti svih podataka sa spoljnih izvora
  - 2. Provera vrednosti podataka prosleđenih od strane drugog dela istog programa
  - 3. Odlučiti kako da se obrađuje loš ulaz

Assert

- Deo koda koji omogućava programu da se proverava u toku izvršavanja.
- Posebno su korisno u velikim, kompleksnim programima. Omogućava programeru da brže nadje greške nastale prilikom modifikacije programa.
- Ključna reč **assert**.
- Obično ima dva argumenta :
  - Uslov koji opisuje pretpostavku koja bi trebala da je tačna.
  - Poruku ukoliko uslov nije ispunjen.

Assert

- Assert proverava uslove koji ne bi trebalo nikako da se dogode.
- Neki od primera korišćenja Assert :
  - Uneta promenjiva upada u očekivani opseg.
  - Pokazivač je non-NULL.
  - Fajl je otvoren za čitanje(pri čitanju fajla)
  - Itd...

Assert

- Primeri :
- **Java:**

```
assert denominator != 0 : "denominator is unexpectedly equal to 0.";
```

- **C++ Assert Macro:**

```
#define ASSERT( condition, message ) {
if ( !(condition) ) {
fprintf( stderr, "Assertion %s failed: %s\n", #condition, message );
exit( EXIT_FAILURE );
}
                                                    }
```

Assert

- 1.Izbegavati izvršni kod u assert:
- Loš primer :
  - Debug.Assert(PerformAction()) ' Couldn't perform action
- Ispravka :
  - actionPerformed = PerformAction()

    Debug.Assert(actionPerformed) ' Couldn't perform action
- 2. Koristiti assert za dokumentovanje preduslova.
- 3. Za robustan kod, assert, a nakon toga i obraditi grešku

Tehnike obrade grešaka

- Assert se koristi za obradu grešaka koje nikako ne bi trebalo da se dese. Ostale greške koje su moguće, očekivane treba obraditi na neki od sledećih načina :

Tehnike obrade grešaka

1. Vraćanje neutralne vrednosti
2. Zamena za sledeći dobar podatak
3. Vraćanje ogovora koji je dobijen za prethodnu ispravnu vrednost podatka
4. Zamena za najbližu validnu vrednost
5. Prikačivanje upozorenja
6. Vraćanje koda greške
7. Pozivanje mehanizma za obradu greške
8. Prikazivanje greške kad god se na nju naleti
9. Gašenje sistema

Tehnike obrade grešaka

- Ispravnost nikada ne vraćati neispravne vrednosti.
- Robusnost uvek pokušati da se nađe način da se program izvrši.
- Robusnost vs Ispravnost.
- Zbog mnogo opcija prilikom obrade grešaka voditi računa o konzistentnosti.

Izuzeci

Atribut Izuzetka	C++	Java	Visual Basic
try-catch	da	da	da
try-catch-finally	ne	da	da
Vrste izuzetaka	Exception objekat, pokazivač ili referenca na objekat. Neki tipovi podataka	Exception objekat	Exception objekat
Efekti neuhvaćenih izuzetaka	std::unexpected()	Gasi thread koji izvršava	Gasi program
Izuzeci koji se bacuju moraju biti definisani u class interface	ne	da	ne
Izuzeci koji se hvataju moraju biti definisani u class interface	ne	da	ne

Izuzeci

- Koristiti izuzetke kao obaveštenja drugim delovima programa o grešci koja se ne bi trebalo ignorisati.
- Vratiti izuzetke samo za uslove koji su stvarno izuzetni tj. ne mogu se drugačije uhvatiti.
- Izbegavati izuzetke u konstruktorima i destruktorima, ukoliko se ne hvataju u istim
- Bacanje izuzetaka na odgovrajućem nivou apstrakcije
- Sve informacije koje su dovele do izuzetka uneti u poruku o izuzetku
- Izbegavati prazne catch blokove
- Poznavanje izuzetaka koje baca korišćena biblioteka
- ReportException
- Standarizovano korišćenje izuzetaka

Ograđivanje štete u programu izazvane greškama u kodu

- Firewall
- ″Bezbedni ″ delovi programa. Sve greške unutar bezbednog dela programa ″assert″-ovati.
- Jedan deo softvera može da radi sa lošim podacima, drugi ne.
- Prebacivanje unetih podataka na ispravne vrednosti prilikom unosa.

Debugging Aids

- Razlika između završne i razvojne verzije softvera.
- Zamena brzine i resursa tokom razvoja za ugrađene alate koji mogu da olakšaju razvijanje softvera.

Debugging Aids

- Što ranije uvođenje u kodu.
- "Ofensivno" programiranje :
  - Assert gasi program.
  - Namerno izazivanje alokacijskih greškaka.
  - Namerno izazivanje grešaka u radu sa tokovima podataka.
  - Itd...
- http://johannesbrodwall.com/2013/09/25/offensive-programming/

Debugging Aids

- Planiranje uklanjanja Debugging Aids
  - Korišćenje Version Control alata.
  - Korišćenje ugrađenog pretprocesora.
  - Pisanje pretprocesora.
  - Smanjenje debug postupaka u završnoj verziji

```
#define DEBUG
    #if defined( DEBUG )
    #define DebugCode( code_fragment ) { code_fragment }
    #else
    #define DebugCode( code_fragment )
    #endif
    DebugCode(
    statement 1;
    statement 2;
    statement n;
    );
```

https://www.atlassian.com/dvcs/overview/what-is-version-control

Koliko odbrambenog programiranja ostaviti u završnom kodu

- Ostaviti kod koji proverava važne greške.
- Brisati deo koda koji proverava greške sa trivijalnim posledicama.
- Brisanje koda koji rezultira gašenjem programa, i pritom ne omogućava čuvanje rada.
- Log grešaka

Sumiranje

- U završnoj verziji programa nema pristupa ″garbage in - garbage out″.
- Tehnike odbrambenog programiranja omogućavaju lakše pronalaženje grešaka, njihovo lakše ispravljanje i smanjuje njihovu štetu.
- Assert omogućava rano pronalaženje greške u izvršavanju.
- Jako je važna odluka kako se nositi sa lošim unosom.
- Izuzeci
- Razlika između završne i razvojne verzije softvera.