CppUnit Tutorijal za Windows

Sadržaj:

1. Kako napraviti CppUnit projekat

1) Napraviti praznu Win32 Console Application, npr. Test (Project name)

2) Izabrati empty project

3) Otvoriti Project->Properties, unutar Properties otvoriti Configuration Properties

3.1) Otvoriti C/C++ deo

3.1.1) Otvoriti Code Generation deo

3.1.1.1) Izabrati run-time library: Multi-threaded Debug DLL (/MDd)

3.1.2) Otvoriti Precompiled Headers deo

3.1.2.1) Izabrati Not using precompiled headers

3.1.3) Otvoriti General deo

3.1.3.1) U Additional Include Directories uneti include direktorijum od CppUnit, npr. d:\D\Programming\CppUnit\cppunit-1.12.1\include

3.2) Otvoriti Linker deo

3.2.1) Otvoriti Input deo

3.2.1.1) U polje Additional Dependencies dodati: cppunitd.lib

3.2.2) Otvoriti General deo

3.2.2.1) U Additional Library Directories dodati lib direktorijum od CppUnit, npr. d:\D\Programming\CppUnit\cppunit-1.12.1\lib

4) Ponovo prevesti projekte u okviru CppUnit (u sustini je dovoljan projekat cppunit_dll), ali sa opcijom Enable C++ exceptions (/EHa) u Project Properties->Configuration Properties->Code Generation delu

5) Otvori New, izabrati C++ Source File, uneti File name (e.g. Main), kliknuti OK, i uneti tekst programa (npr. sadržaj Main.cpp iz originalnog CppUnit primera "simple")

6) Otvoriti Build, izabrati Build Test; Rezultat bi trebao da bude uspesan.

7) Otvoriti Debug, izabrati Start Without Debugging

8) Na ekranu treba da se dobije tekst "OK(0)"

2. Objašnjenje Main.cpp iz originalnog CppUnit primer Simple

- Ovo je generička datoteka u kojoj se formira CppUnit okruženje za automatsko registrovanje, izvršenje i izveštavanje o rezultatima svih CppUnit testova, koji su definisani u projektu.
- Kao takva može se, i poželjno je, koristiti je u svim ispitnim zadacima, čija verifikacija se zasniva na CppUnit.
- Okruženje se sastoji od:
    - Test kontrolora (objekat `controller`),
    - Skupljača rezultata (objekat `result`),
    - Nadzornika posla (objekat `progress`),
    - Izvršioca skupa testova (objekat `runner`), i
    - Izveštača o rezultatima (objekat `outputter`).

3. Objašnjenje ExampleTestCase.h iz primera Simple

- Uvek treba uključiti pomoćne makroe definisane zaglavljem HelperMacros.h.
- Klasa `ExampleTestCase` predstavlja primer grupe testnih slučajeva (ne pojedinačnog slučaja, već grupe slučajeva).
- Sve ovakve klase se izvode iz CppUnit klase `TestFixture`.
- Makro `CPPUNIT_TEST_SUITE` označava početak deklaracije grupe slučajeva, a makro `CPPUNIT_TEST_SUITE_END` kraj te deklaracije.
- Pomoću makroa `CPPUNIT_TEST` se deklariše pojedinačni testni slučaj, koji je predstavljen metodom (funkcijom članicom) klase koja predstavlja grupu slučajeva. Prva takva metoda je metoda `example`.
- Slede deklaracije objekata (eng. test fixture) nad kojima rade testni slučajevi. U ovom primeru to su `m_value1` i `m_value2`.
- Zatim se deklarišu metoda za pravljenje objekata nad kojim rade testni slučajevi, `setUp()` i metode za uništavanje tih objekata `tearDown()`. U ovom primeru je deklarisana samo `setUp()`.
- Na kraju su dati prototipovi testnih metoda, koje moraju biti zaštićene (protected).

4. Objašnjenje ExampleTestCase.cpp iz primer Simple

- Na početku se uključuje zaglavlje ove klase ExampeTestCase.h.
- Zatim se pomoću makroa `CPPUNIT_TEST_SUITE_REGISTRATION` registruju svi testni slučajevi (testne metode) u ovoj klasi. Ovom registracijom je omogućeno kasnije automatsko izvršenje i izveštavanje o rezulatima izvršenja registrovanih testnih slučajeva.
- Dalje sledi definicija `setUp()` metode i pojedinačnih test metoda `example()`, itd.
- U testnim metodama (slučajevima) se pomoću makroa `ASSERT` proveravju određeni zadati uslovi koji moraju biti ispunjeni u određenim fazama izvršenja testnog slučaja. Ako je uslov ispunjen, nastavlja se normalno izvršenje programa. U suprotnom slučaju, pravi se i izdaje odgovarajući izveštaj o grešci. Sve ide potpuno automatski.
- Test slučajevi mogu biti uspešno završeni (OK), neuspešno (Failure) i sa greškom (Error). Slučaj postaje neuspešan (Failure) čim se naiđe na prvi ASSERT koji nije zadovoljen. Ukoliko u toku izvršenja slučaja dođe do izuzeća (npr. deljenje sa 0), taj testni slučaj se završava sa greškom (Error), i automatski se prelazi na izvršenje sledećeg registrovanog testnog slučaja.

5. Objašnjenje CppUnit testa za rešenje zadatka iz I vežbe pomoću semafora

- Uvedena je klasa Vezba_2aa koja predstavlja grupu testnih slučajeva sa jednim testnim slučajem, tj. metodom, `test1()`.
- Bivša funkcija main() iz modula vezba_2aa.cpp je rasparčana na sledeće funkcije članice (metode) nove klase: `setUp()`, `test1()` i `tearDown()`. U ovim metodama je ubačen pozdravni ispis iz edukativnih razloga. U normalnoj upotrebi takve ispise izbegavamo kao suvišne i nepotrebne.
- Pošto Win32 API originalno predviđa da funkcija koju izvršava dinamički napravljena nit bude globalna C funkcija, funkcija `print()` je ostala takva (kakva je i bila).
- Da bi se omogućila saradnja funkcije `print()` i funkcija članica nove klase, svi podaci koji se odnose na semafore i niti su globalni.
- Za potrebe testiranja i verifikacije dodat je sistemski log/trag (`string log`).
- Provera ispravnosti rešenja se sada obavlja putem poziva makroa `CPPUNIT_ASSERT_EQUAL` u funkciji `test1()`, gde se prvim parametrom zadaje očekivani trag sistema, a u drugom se prosleđuje stvarni trag koji se proverava.

6. Objašnjenje CppUnit testa za primer proizvođač-potrošač iz II vežbe

- Prilagođenje modula vezba_2b CppUnit okruženju je urađeno na jednostavniji način nego u slučaju vezba_2aa (prethodna tačka 5.). Ovde je jednostavno uvena nova klasa `Vezba_2b` i preimenovana je funkcija `main()` u `Vezba_2b::test1()`. To je dovoljno.

- Naravno, ovaj pristup ima problem izolacije pojedinih testnih slučajeva, zato što nema `setUp()` i `tearDown()` metoda.
- Uvedena su dva traga, jedan za proizvođača i jedan za potrošača. Provera ispravnosti rešenja u test metodi `test1()` se zasniva na činjenici da ova dva traga moraju biti identična.
- Inače, treba da se trudimo da testne slučajeve pišemo pre (TDD) i u toku razvoja programske podrške, tako da već od početka istestiramo najpre sve osnovne mehanizme (primer je test metoda `test2`), pa pojedinačne module, a tek na kraju celo rešenje. Svi ovi testovi se mogu realizovati uz pomoć CppUnit.
- Test metoda `test2()` je primer jednostavnog testa osnovnih mehanizama – smeštanje i preuzimanje znakova iz kružnog bafera.
- **Uputstvo za ispitne zadatke br. 1**: pokušajte da najpre napravite pojedinačne celine i da njih pojedinačno istestirate sa CppUnit testovima. Npr. `test3()` pojedinačno testira isključivo potrošača, dok sam test igra ulogu proizvođača. U ovoj test metodi, testni slučaj najpre napravi samo potrošača, i naravno kružni bafer sa svom pratećom signalizacijom (semafori i kritična sekcija). Zatim obriše tragove proizvođača i potrošača, stavi 8 znakova u kružni bafer, sačeka da ih potrošač potorši, a zatim proveri njegov trag.
- **Uputstvo za ispitne radove br. 2**: Treba da unapred razmišljamo kako da obezbedimo uslove za testiranje (D4T ili design for testability). Jedan primer toga su tragovi proizvođača i potrošača u ovom primeru. Testovi se pišu „spolja“, tj. objekat testiranja (programski kod koji je predmet testiranja) ne smemo menjati, dodavanjem npr. nekih kontrolnih ispisa. Uvek treba da razmišljamo kako da programski pobuđujemo ulaz i kako da programski prihvatamo izlaz iz jedinice koju testiramo (eng. test harness), tako stvaramo uslove za potpuno automatsko testiranje.

7. Objašnjenje CppUnit testa za primer mešanja ispisa 3 stringa iz I vežbe

- Modul vezba_1b_v2.cpp je uključen tako što je dodata nova klasa Vezba_1b_v2. Bivša funkcija `main()` je sad postala `test1()`. Ovaj put su dodate i funkcije `setUp()` i `tearDown()`.
- Slično kao i u prethodna 2 primera, dodat je sistemski trag. Svaka nit u njega unosi svoj redni broj pre ispisa i nakon ispisa. Ukoliko nema sinhronizacije, ovi brojevi će biti pomešani. Ukoliko je pak ima, svaka 2 susedna broja u tragu, počev od prvog, moraju biti jednaka.
- Rešenje se lako može proveriti ako se recimo zakomentariše ulazak i izlazak iz kritične sekcije u programskom kodu prve niti.