Kodavimo stilius

Išeities teksto formatavimas

1. Nerašykite eilučių, ilgesnių negu **80 simbolių**. Ilgas eilutes sulankstykite ties matematiniais operatoriais, priskyrimo operatoriais, funkcijos argumentais.
2. Naudokite 4 pozicijas blokų atitraukimui.
3. Naudokite tarpo simbolius, o ne tabuliacijas, atitraukti bloko operatoriams.
4. Nerašykite tarpo prieš atidarantį skliaustelį „(“ valdymo struktūrose ir funkcijų kvietimuose. Visada dėkite tarpą po atidarančio skliaustelio ir prieš uždarantį skliaustelį, jei jis neina po kito uždarančio skliaustelio („mano stilius“?).
5. Naudokite tarpus išskirti aritmetiniams operatoriams, jei išraiškos telpa į vieną eilutę. Jei išraiškos ilgesnės kaip viena eilutė, nuosekliai ištrinkite tarpus aplink visus aukščiausio prioriteto operatorius; jei to neužtenka, kad išraiška tilptų į eilutę – aplink žemesnio prioriteto operatorius ir t.t. Tuo būdu išraiška vizualiai suskaldoma į, pvz., sandaugų sumą.
6. Naudokite Perl'o stilių riestiniams blokus žymintiems skliausteliams „{“ ir „}“ išdėstyti; panašų stilių naudokite ir C kodui (pvz. K&R stilių su skliausteliais net ir blokui iš vieno operatoriaus, arba 1TBS stilių).
7. Aš visada rašau C funkcijos grąžinamos reikšmės tipą toje pačioje eilutėje, kaip ir funkcijos vardas, bet GNU kodo standartai siūlo rašyti funkcijos grąžinamos reikšmės tipą atskiroje eilutėje, kad funkcijos vardą galima būtų pradėti nuo 1 stulpelio. Abu stiliai geri, bet viename projekte reikia nuosekliai naudoti vieną stilių, koks jis bebūtų.
8. Visais nepaminėtais atvejais reikia naudoti GNU kodavimo standartus (http://www.gnu.org/prep/standards/standards.html) and the Gnome coding style (http://developer.gnome.org/doc/guides/programming-guidelines/code-style.html).

Vardai

1. Nenaudokite santrumpų kintamųjų vardams – 'valandos' arba 'value' yra žymiai geresnis vardas negu 'val'. Naudokite tik „gerai žinomas santrumpas“ (pvz. x, y, z ortogonaliosioms koordinatėms), sutartas projekte, o savo naujus sutrumpinimus dokumentuokite komentaruose ir naudokite nuosekliai. Savus sutrumpinimus įveskite tik tada, kai vardas labai dažnai kartojasi sudėtingose išraiškose ir sutrumpinimo įvedimas palengvins kodo skaitymą.
2. Naudokite paaiškinančius prasmę kintamųjų vardus; „$average_coordinate“ yra geresnis vardas negu '$val' ar gink dieve '$tempry'.
3. Naudokite paaiškinančius funkcijų vardus, kurie aprašo, ką funkcija turi atlikti.
4. Kalbose kurios neturi vardų galiojimo sričių, tokiose kaip senoji C,

naudokite paketo vardą visų eksportuojamų vardų prefiksuose.

Klaidų pranešimai

1. Naudokite mūsų laboratorijos vidinį standartą klaidų pranešimų formatavimui. Iš esmės tai GNU pranešimų stilius su nedideliais pakeitimais, kad būtų lengviau klaidų pranešimus apdoroti automatiškai.
2. Klaidų pranešimuose arba perspėjimuose visada turi būti nurodyta:
    1. programos, kurią naudotojas pakvietė, vardas (bet, kaip taisyklė, paprogramės vardas, kurioje įvyko klaida, nereikalingas),
    2. failo, kuris buvo apdorojamas, kai įvyko klaida, vardas,
    3. pozicija faile, kur buvo aptikta klaida (eilutė, pozicija eilutėje, duomenų bloko ar kito skirsnio vardas),
    4. trumpas bet suprantamas klaidos aprašymas, pranešantis, kas atsitiko ir kaip klaidą galima būtų pašalinti,
    5. jei klaida atsitiko, vykdant sistemos bibliotekos funkcijas, reikia įterpti ir sistemos klaidos pranešimą.
3. Naudokite vieną funkciją klaidų pranešimams suformuoti.

Komentarai

1. Prieš kiekvieną funkciją reikia parašyti vieno-dviejų sakinių komentarą, paaiškinantį, ką funkcija turi daryti, ir išvardijantį specialius reikalavimus, kurie nėra akivaizdūs iš funkcijos signatūros. Pavyzdžiui, jei C funkcija grąžina char* simbolių eilutę, reikia nurodyti, ar ši eilutė statinė, ar atmintis jai paskirta kokia nors *alloc() šeimos funkcija ir turi būti grąžinta į laisvos atminties sąrašą funkcija free().
2. Nekomentuokite akivaizdžių dalykų. Labai blogo komentaro pavyzdys yra: 'int i = 0; /* kintamasis i inicializuojamas nuliu */'. Kodas šiuo atveju trumpesnis ir aiškesnis negu komentaras. Be to, komentaras taps neteisingu, jei kada nors inicializaciją teks pakeisti į 'int i = 1', ir tokios klaidos nepagaus joks kompiliatorius.

   Kitaip tariant, **komentarai turi būti ortogonalūs kodui**.
3. Komentarai turėtų aprašyti ne tai, ką kodas daro, bet tai, ką jis turi daryti.
4. Atminkite, kad kompiliatorius komentarų teisingumo netikrina, ir greičiausiai, kai kodas bus keičiamas, programuotojai pamirš pakeisti komentarus, ypač esančius toliau nuo keičiamos kodo vietos. Todėl geriausia, kad komentarai aprašytų bendrus, nesikeičiančius kodo dėsningumus, o ne dabartinės realizacijos smulkmenas. Statiškai tipizuotoms kalboms neaprašinėkite funkcijų parametrų tipų (nors tai būtina dinamiškai tipizuotoms ir netipizuotoms kalboms). Geriau aprašykite funkcijų argumentų „fizikinę prasmę“, kuri nesikeis (greičiausiai dėl to, kad pakeitus ją, nustos veikti per daug programos dalių).
5. Komentarai tikriausiai išliks po to, kai kodas bus pakeistas, todėl bandykite suformuluoti komentarus, nurodydami kontekstą, kuriame jie

rašomi. Pvz., komentaras „funkcijos parametras 'molecule' negali būti lygus NULL“ taps klaidingu, kai funkcija bus pakeista taip, kad galėtų apdoroti parametrą lygų NULL. Tačiau teiginys „funkcijos versija 1.1 negali apdoroti parametro 'molecule' lygaus NULL“ liks teisinga net tada, kai funkcija versijoje 2.0 bus patobulinta ir galės apdoroti tokį parametrą.

Vienas tiesos šaltinis; kodo perrašymas

1. Niekada nekartokite tos pačios informacijos keliose vietose. Infomacija (kintamasis, konstanta, kodas, paprogramė), reikalinga keliuose programos taškuose, turi būti sudėta į atskirą modulį, ir visi naudotojai turi ją iš ten pasiimti.
2. **Jei jums kyla noras nukopijuoti funkciją į kitu vardu ir truputį ją pakeisti, niekuomet to nedarykite!** Vietoj to, perrašykite kodą, kad ta pati funkcija būtų naudojama visose vietose, kur to reikia. Tai galima įgyvendinti keliais būdais:
    1. jei reikalinga bendresnė funkcija su platesniu interfeisu, paverskite seną funkciją bendresne, pavadindami ją nauju vardu, o senąją funkciją realizuokite kaip naujosios kvietimą, gal būt nurodant tam tikras fiksuotas parametrų reikšmes;
    2. jei patogumo dėlei reikalinga panaši funkcija su kiek kitokiu interfeisu, realizuokite bendresnę funkciją, tinkamą abiem atvejam, ir abu dalinius interfeisus realizuokite kaip funkcijas, kviečiančias bendrąją, sudėtingąją funkciją;
    3. jei naujajai funkcijai reikia pakeisti nedidelį fragmentą senosios funkcijos kodo viduryje, gal būt galima suskaldyti senąją funkciją į tris pagalbines, kurių dvi (pirmoji ir paskutinioji) būtų bendros abiem reikalingom funkcijom ir pakviečiamos iš jų.
3. **Jei kodo perrašymas pakeičia funkcijos elgesį, būtinai pakeiskite funkcijos vardą.** Jei svarbu palikti seną funkcijos vardą, pirmą realizuokite naują funkciją nauju vardu, pakeiskite visus jos kvietimus, įsitikinkite, kad senosios funkcijos kvietimų kode neliko, o po to pervadinkite naują funkciją senuoju vardu. Tai ypač svarbu, jei pasikeičia tie funkcijos interfeiso aspektai, kurių kompiliatorius negali patikrinti automatiškai.

Funkcijos, parametrai ir kintamieji

1. **Globalūs kintamieji yra blogis.** Venkite jų kiek įmanoma. Visi funkcijų darbui reikalingi duomenys turi būti perduodami funkcijoms kaip parametrai. Jei reikia perduoti labai daug parametrų, naudokite struktūras arba objektus. Net jei reikia įvesti papildomą parametrą daugeliui funkcijų kvietimų grandinėje, geriau tai padaryti, arba realizuoti naują interfeisą, paliekant senąjį, negu įvedinėti globalius kintamuosius. Globalūs kintamieji:
    1. labai apsunkina kodo lydėjimą;
    2. padaro kodą nereenterabilų.

    Vienintelis atvejas, kai globalūs kintamieji yra pakankamai saugūs ir

pateisinami, yra 'debug' vėliavėlės (angl. flags).

2. Idealiu atveju, paprogramė turi elgtis kaip funkcija, t.y. nemodifikuoti savo parametrų, o visus rezultatus grąžinti kaip gražinamas reikšmes. Jei naudojama programavimo kalba leidžia grąžinti iš paprogramės daug reikšmių, naudokite šią galimybę, o ne modifikuokite 'var' klasės paprogramės parametrą.
3. Venkite parametrų, kurie praneša paprogramei, kokią funkciją ji turi vykdyti (pvz., nenaudokite tokių parametrų: 'change(int x, in task) /* task == 1 – pakeisti spalvą, task == 2 – pakeisti formą */'. Jei eisite šio keliu, Jūsų programa gali pavirsti keistu netipizuotų baitų interpretatoriumi :). Geriau sukurkite dvi nepriklausomas funkcijas 'change_colour(int x)' ir 'change_shape(int x)'.
4. Neįsiūkite sprendimų apie bendrą skaičiavimų „politiką“ į žemo lygio paprogrames (įrankius). Jei žemo lygio paprogramė turi žinoti, kokia naudojama skaičiavimo politiką (pvz., kur galima sukurti ir ar galima sukurti laikinus failus, ar grąžinti derinimui reikalingą informaciją), geriau perduokite jai parametrą, nurodantį, kokią politiką pasirinko aukšto lygio, valdantis modulis, arba gražinkite papildomą reikšmę, kad šaukiantysis modulis galėtų nuspręsti, kokios politikos laikytis.

Teiginių patikrinimai (asserts)

1. Naudokite teiginių, kurie turi būti teisingi, patikrinimus, kad užtikrintumėte funkcijų darbui būtinas pradines sąlygas. **Visada** naudokite assert() patikrinimus, jei neišpildžius sąlygos programa pasibaigs avariniu būdu (pavyzdžiui, sukeldama „segmentation fault“ arba „bus error“ išimtinę situaciją). Pavyzdžiui, C kalboje naudokite assert(p != NULL) arba assert(p) prieš kreipdamiesi į p->val lauką, jei, žinoma, ši sąlyga nėra akivaizdžiai užtikrinta if() arba while() sąlygos.
2. Perl'o kalboje naudokite assert() iš Carp::Assert paketo arba tiesiog 'die unless ...'. Naudokite juos visada tose vietose, kur programa veiktų neteisingai arba negalėtų tęsti darbo, jei tikrinama sąlygą neišpildyta, bet kur Jūs manote, kad sąlyga turi būti teisinga nepriklausimai nuo apdorojamų duomenų ar įvykių.