Programmeerimisest üldiselt

Programmeerimine tähendab käskude jada ehk programmi koostamist arvutile täitmiseks. Programm võib olla kirja pandud erinevates programmeerimiskeeltes. Madaltaseme keeled (näiteks assembler) on üldiselt ebamugavamad kasutada, aga nendes keeltes kirjutatud programmid töötavad tihti kiiremini. Kõrgtaseme keelte (näiteks Python) kasutamine on lihtsam, aga nendes keeltes kirjutatud programmid töötavad üldiselt aeglasemalt.

Programm on eripärane toode, kuna see ei vaja füüsilisi toormaterjale ja kui üks eksemplar on valmis, siis selle kopeerimine on triviaalne – erinevalt autodest, teleritest, rõivastest ja praktiliselt kõigist muudest toodetest. Põhiressurss ja põhikulu programmi loomise juures on tarkvaraarendajate töö: nende võime probleemist aru saada, lahendus välja mõelda ja programmi keerukust hallata. Siit ka vajadus kõrgtaseme keelte järele, mis seda ressurssi kokku hoiavad. See on ka põhjus, miks tarkvara sisaldab üldiselt rohkem vigu kui muud tooted: arendustöö on siin praktiliselt kogu kulu.

Keele süntaksi tundmine on programmeerimiseks tarvilik, aga mitte piisav tingimus. Nii nagu klaverikontserdi komponeerimiseks ei piisa teadmisest, et klahvidele vajutamine põhjustab helisid nii on ka vähegi keerukama programmi koostamiseks vaja lisaks keele käskude tundmisele ka võimet neid käske kokku sobitada. See eeldab tõsist harjutamist. Programmeerimine on valdkond, kus oskuste vahed on väga selged. Vahe professionaalsete programmeerijate produktiivsuses võib olla 10:1 [1].

1. Programmeerimine kui probleemilahendamine

Programm on lahendus mingile probleemile. Programm ei ole iseenesest õige või vale vaid see on õige või vale mingi spetsifikatsiooni suhtes.

Järgmised probleemilahendamise sammud on inspireeritud G. Polya raamatust „Kuidas seda lahendada?“ [2]. Need sobivad hästi algoritmiliste probleemide lahendamiseks, kus programm peab etteantud sisendi põhjal leidma sobiva väljundi.

1. Probleemist arusaamine (analüüs)
   - Mis on otsitav väljund? Mis on sisend? Milles seisnevad ülesande tingimused?
   - Kas väljund on sisendi põhjal üldse leitav?
   - Võtke kasutusele sobiv terminoloogia. Valmistage joonis.

2. Algoritmi koostamine
   - Kas olete sellist ülesannet varem näinud? Või olete näinud sarnast ülesannet veidi teisel kujul? Kas siin saab kasutada standardteeke?

   - Kui olete leidnud sarnase tuntud ülesande, siis kas saate selle lahendust otse kasutada? Või on vaja tuua sisse mõni abielement või andmeid teisendada?

- Kas ülesande juures saab eristada selgelt defineeritud alamülesandeid? Defineerige need alamülesanded ja proovige leida nende jaoks sobivad algoritmid. Defineerige vastavad funktsioonid.
- Vaadelge otsitavat väljundit. Kas mõnes tuntud ülesandes on olnud vaja leida sarnast väljundit?
- Kas ülesannet saab teisiti sõnastada?
- Kui te ei suuda ülesannet lahendada, siis proovige kõigepealt lahendada mõni temaga seostatav ülesanne. Kas saate koostada antuga seostatava, kuid jõukohasema ülesande? Üldisema ülesande? Erijuhu?

3. Tagasivaade
   - Kas saate kontrollida tulemust? Kas lisaks tüüpjuhule olete testinud ka erijuhtusid (lihtsaim või suurim võimalik sisend, muud erijuhud)?
   - Kas saate väljundit teisiti või lihtsamalt leida?
   - Kas algoritmi kiirus või mälukasutus on kriitilised? Kas algoritmi on võimalik optimeerida?
   - Kas seda algoritmi on lihtne korduvkasutada? Kas kood on korralikult dokumenteeritud ja kommenteeritud? Kas funktsioonide ja parameetrite nimed on sisulised ja arusaadavad?

Keeruka probleemi jagamine alamprobleemideks on keerukate tarkvarasüsteemide juures kriitiline oskus. Selle toetamiseks on programmeerimiskeeltes palju mehhanisme: funktsioonid, klassid, moodulid.

Sobivate väliste teekide (tarkvaramoodulite) leidmine ja kasutamine on veel üks kriitiline programmeerimisoskus. Pythoniga tuleb kaasa hulk standardteeke [3]. Nagu Pythoni kogukonnas öeldakse on tegemist *batteries-included* keelega.

Algoritmi kiiruse ja mälukasutuse optimeerimine on suur ja palju-uuritud valdkond. On olemas algoritmide keerukusteooria, suur hulk algoritme väga erinevate probleemide lahendamiseks ja iga aasta ilmub hulgaliselt teadusartikleid algoritmika valdkonnast.

2. Programmeerimine kui kommunikatsioon

Programmeerimine on mitte ainult kommunikatsioon arvutiga vaid tihti ka kommunikatsioon teise programmeerijaga, kes soovib teie programmi mõista, muuta või korduvkasutada. Teiseks programmeerijaks võite olle teie ise mõni kuu või aasta pärast programmi kirjutamist. Probleemi tükeldamine selgelt defineeritud alamprobleemideks, mis on ka oluline probleemilahendamise oskus, muudab programmi arusaadavaks. Samuti teeb seda sobiva terminoloogia kasutamine moodulite, funktsioonide ja parameetrite nimedes. Probleemi selge mõistmine ja selgelt kirja pandud ja struktureeritud programm käivad käsikäes.

Kasutatud kirjandus

[1] DeMarco, T. Lister, T. Peopleware, 2nd ed (1999).

[2] Polya, G. Kuidas seda lahendada? (2001)

[3] The Python Standard Library, http://docs.python.org/2/library/ (WWW, 30.01.2014)