Imperatiivisten korkean tason ohjelmointikielten sukupuu

Turun yliopisto
Tietotekniikan laitos
LuK-tutkielma
Marraskuu 2024
Timo Rintamäki

TURUN YLIOPISTO
Tietotekniikan laitos

Timo Rintamäki: Imperatiivisten korkean tason ohjelmointikielten sukupuu

LuK-tutkielma, 29 s.
Marraskuu 2024

Suosituimmat käytössä olevat korkean tason ohjelmointikielet noudattavat imperatiivista paradigmaa. Imperatiivisia ohjelmointikieliä on vuosien varrella kehitetty sekä uusia luoden että vanhoja päivittäen. Uudet ohjelmointikielet eivät kuitenkaan ilmesty tyhjästä, vaan ne perustuvat osittain tai kokonaan aikaisemmin julkaistuihin ohjelmointikieliin. Päällepäin ei ole aina ilmiselvää, mihin muihin ohjelmointikieliin jokin tietty ohjelmointikieli perustuu.

Tässä tutkielmassa tarkastelun kohteena on valikoitujen imperatiivisten ohjelmointikielten sukulaissuhteet, jotka voidaan päätellä niiden kehityksessä inspiraationa toimineista ohjelmointikielistä. Lisäksi tarkastelussa ovat syyt ohjelmointikielten suurelle lukumäärälle. Asiaa lähestytään kirjallisuuskatsauksena ohjelmointikielten varhaisiin dokumentaatioihin sekä kyseisten ohjelmointikielten kehittäjien kirjoittamiin artikkeleihin ja heistä tehtyihin haastatteluihin. Selvityksen perusteella ohjelmointikielille muodostetaan sukupuu, josta on nähtävissä olennaisimmat yhteydet ohjelmointikielten välillä.

Tutkielmassa esiintyvien ohjelmointikielten kehitys on perustunut aluksi tietokoneiden ohjelmoinnin helpottamiseen, sillä konekielellä ja symbolisella konekielellä (assembly) ohjelmoimisen katsottiin olevan hidasta ja vaivalloista. Myöhempien ohjelmointikielten kehitystä on ohjannut pyrkimys luoda tiettyihin käyttökohteisiin paremmin soveltuvia työkaluja. Tällaisia käyttökohteita ovat esimerkiksi kaupalliset sovellukset, simulaatioiden ohjelmointi, ohjelmistojen rakenteen selkeyttäminen olio-ohjelmoinnilla, järjestelmäohjelmointi ja web-ohjelmointi. Muodostetusta sukupuusta havaitaan, että imperatiivisten ohjelmointikielten juuret ovat Fortranissa sekä ALGOL-perheen ohjelmointikielissä. Selvää on myös, että uusia ohjelmointikieliä ei ensimmäisiä lukuun ottamatta luoda tyhjästä, vaan sukupuusta on havaittavissa selkeä jatkumo vanhimmista ohjelmointikielistä uusimpiin.

Asiasanat: imperatiivinen ohjelmointi, ohjelmointikielten sukupuu

Sisällys

1 Johdanto

Ohjelmointi koskettaa ihmisten elämää lähes jokaisella elämän osa-alueella. Tietokoneita — ja niiden sisältämiä ohjelmia — on kodinkoneissa, liikenteessä, puhelimissa, älykelloissa ja muissa sähkölaitteissa. Jotta tietokoneet tekisivät mitä niiltä odotetaan, on niille kirjoitettava ohjeet; nämä ohjeet muodostuvat biteistä eli ykkösistä ja nollista. Tietokone ymmärtää bittijonoina annettuja ohjeita, mutta ihmiselle bittijonojen lukeminen ja niiden kirjoittaminen ei ole helppoa saati nopeaa. Ensimmäisten tietokoneiden ohjelmoinnin varhaisessa vaiheessa niiden parissa työskennelleet ihmiset alkoivat kehittää helpompia tapoja kirjoittaa ohjelmia. Näin saivat ennen pitkää alkunsa korkean tason ohjelmointikielet, joilla voidaan kirjoittaa ihmisille helpommin ymmärrettävää ohjelmakoodia. Tietokone ei kuitenkaan suoraan ymmärrä korkean tason ohjelmointikieltä sellaisenaan, vaan lähdekoodi käännetään biteiksi eli konekieleksi, kun ohjelma halutaan suorittaa.

Eri ohjelmointiparadigmoja on useita: muun muassa imperatiivinen ohjelmointi, deklaratiivinen ohjelmointi, funktionaalinen ohjelmointi, proseduraalinen ohjelmointi ja olio-ohjelmointi. Yksittäinen ohjelmointikieli voi noudattaa useampaa paradigmaa samanaikaisesti, esimerkiksi imperatiivista sekä olio-ohjelmointia. Imperatiivisessa ohjelmoinnissa annetaan askel kerrallaan suoritettavia käskyjä, jotka voivat muokata ohjelman tilaa. Statistan ohjelmointikielten suosiota mittaavassa kyselytutkimuksessa suosituimmat ohjelmointikielet ovat imperatiivista paradigmaa noudattavia kieliä [1].

1.1 Tutkimuskysymykset

Tutkielma käsittelee imperatiivisia ohjelmointikieliä ja niiden kehityskulkua. Tutkielman tavoitteena on vastata seuraaviin kysymyksiin:

- Mitkä asiat ovat ohjanneet imperatiivisten ohjelmointikielten kehitystä?
- Mitkä imperatiiviset ohjelmointikielet voidaan tulkita toistensa sukulaisiksi?

Tutkimuskysymyksiin etsitään vastauksia ohjelmointikielten historiasta, jota tutkimalla voidaan selvittää syyt eri ohjelmointikielten kehittämiseen sekä niiden aikaisemmista ohjelmointikielistä saamat vaikutteet.

1.2 Metodologia

Tutkielma on tehty kirjallisuuskatsauksena, ja sen lähteinä on käytetty ohjelmointikielten alkuperäisiä spesifikaatioita sekä kielten kehittäjien myöhemmin kirjoittamia historiikkeja tai heidän haastattelujaan. Spesifikaatiot ja jälkeenpäin kirjoitetut historialliset perspektiivit eivät tyypillisesti ole vertaisarvioituja, mutta lähtökohtana on, että tietyn ohjelmointikielen ominaisuudet ja kehityskulku on parhaiten tiedossa alkuperäisessä dokumentaatiossa sekä kieltä kehittäneillä henkilöillä. Joukossa on myös vertaisarvioituja artikkeleita sekä alan kirjallisuutta. Lähteitä on etsitty Turun yliopiston kirjaston Volter-tietokannasta sekä Google Scholar -hakupalvelusta. Hakusanoina on pääasiassa toiminut ohjelmointikielen nimi mahdollisesti yhdistettynä sanoihin "history", "specification"tai "evolution". Lisäksi tietoa on haettu hakulauseella "(programming) AND (languages OR history OR evolution)".

Tutkielman toisessa luvussa käsitellään imperatiivisten ohjelmointikielten historiaa ja kehityskulkua. Erityisesti huomiota kiinnitetään ohjelmointikielten kehitykseen vaikuttaneisiin asioihin, kuten kielten käyttökohteisiin sekä aikaisemmin kehitettyjen ohjelmointikielten ominaisuuksien niin sanottuun kopioimiseen. Joistain oh-

jelmointikielistä esitetään lyhyitä koodiesimerkkejä havainnollistamaan kielen tiettyjä ominaisuuksia tai syntaksia. Koodiesimerkit eivät ole kovin monimutkaisia, mutta niiden ymmärtäminen ei myöskään ole välttämätöntä tutkielman lukemista varten. Kolmannessa luvussa muodostetaan toisen luvun havaintojen perusteella imperatiivisten ohjelmointikielten sukupuu. Neljännessä luvussa tehdään päätelmät tutkielman tuloksista ja vastataan tutkimuskysymyksiin.

2 Korkean tason ohjelmointikielten historia

Vuonna 1936 Alan Turing kirjoitti artikkelin, jossa hän kuvailee niin kutsutun automaattisen koneen (engl. *automatic machine*), joka kykenee suorittamaan laskentaa. Kone nimettiin myöhemmin Turingin koneeksi. Turingin kone koostuu loputtomasta nauhasta, lukupäästä, tilarekisteristä sekä käskytaulukosta (engl. *table of instructions*). Kone käsittelee nauhalla yhtä merkkiä kerrallaan; nykyisen tilan sekä käsiteltävänä olevan merkin perusteella se voi kirjoittaa käsiteltävän merkin tilalle toisen merkin, siirtyä yhden merkin vasemmalle tai oikealle sekä siirtyä johonkin toiseen tilaan. Artikkelissa Turing kuvailee myös universaalin koneen (myöhemmin universaali Turingin kone), joka kykenee suorittamaan mille tahansa muulle Turingin koneelle tarkoitetun nauhan ja päätymään samaan lopputulokseen, ja täten laskemaan kaiken, mikä ylipäätään on laskettavissa (engl. *computable*). [2]

Imperatiivistä paradigmaa noudattavat ohjelmointikielet perustuvat Turingin koneen toimintaan. Turingin koneelle vaihtoehtoinen laskennan malli on Turingin mentorin, Alonzo Churchin, samoihin aikoihin kehittämä lambdakalkyyli. Tutkielmassa ei käsitellä lambdakalkyyliä eikä siitä johdettuja ohjelmointikieliä, jotka noudattavat tyypillisesti funktionaalista ohjelmointiparadigmaa, kuin korkeintaan ohimennen.

Turing-täydellisyydellä tarkoitetaan ohjelmointikielten (tai jonkin laitteen tms.) kykyä simuloida mitä tahansa Turingin konetta; jos ohjelmointikielellä on mahdollista suorittaa mitä tahansa universaalin Turingin koneen suorittamaa laskentaa, on kieli Turing-täydellinen. Turing-täydellisyydestä puhuttaessa tyypillisesti sivuutetaan Turingin koneen muistin (nauhan) rajattomuusvaatimus. Ollakseen Turing-täydellinen, ohjelmointikielen pitää kyetä peräkkäisyyteen (rivi kerrallaan järjestyksessä suoritus), valintaan eri operaatioiden välillä (IF-lause) sekä loputtomaan toistoon (silmukat tai rekursio). Käytännössä kaikki merkittävät ohjelmointikielet ovat siis Turing-täydellisiä.

Koska Turing-täydellisyys tarkoittaa mahdollisuutta simuloida mitä tahansa muuta Turingin konetta, voidaan päätellä, että mikä tahansa Turing-täydellisellä ohjelmointikielellä tehty ohjelma voidaan tehdä millä tahansa muullakin Turing-täydellisellä ohjelmointikielellä. Herää siis kysymys, miksi erilaisia ohjelmointikieliä on niin paljon, jos sama lopputulos on saavutettavissa niillä kaikilla.

Seuraavaksi paneudutaan ohjelmointikielten historiaan, johon tutustumalla ohjelmointikielten runsas määrä selittyy. Tutkielmassa esitetyt tapahtumat ovat karkeasti aikajärjestyksessä, mutta on syytä ottaa huomioon, että monen ohjelmointikielen kehitys ensimmäisistä prototyypeistä valmiiseen julkaistuun kieleen kääntäjineen on voinut kestää useita vuosia. Tämän vuoksi joidenkin käsiteltävien ohjelmointikielten kehitys on ollut toistensa kanssa samanaikaista eikä peräkkäistä. Tutkielmassa pyritään käsittelemään ohjelmointikieliä mahdollisimman lähellä niiden ensimmäisiä julkaistuja versioita, jotta myöhemmin kehitettyjen kielten vaikutus ja niistä mahdollisesti kopioidut ominaisuudet eivät sekoittaisi järjestystä.

2.1 1950- ja 1960-luku

Ennen vuotta 1954 suurin osa ohjelmoinnista suoritettiin joko konekielellä tai symbolisella konekielellä (assembly). Joitain yrityksiä ohjelmoinnin helpottamiseksi oli

tehty muun muassa varhaisten kääntäjien kehittämisessä, mutta niiden ongelmana oli tyypillisesti huomattavasti hitaampi suoritusaika verrattuna suoraan konekielellä kirjoitettuun ohjelmaan, mikä johti yleiseen uskomukseen, ettei automaatiolla ole saavutettavissa suorituskykyistä ohjelmakoodia [3]. Muiden epäluuloista huolimatta korkean tason ohjelmointikieliä kuitenkin kehitettiin. 1950-luvulla alkunsa sai tieteelliseen ohjelmointiin tarkoitettu Fortran, jota pidetään ensimmäisenä varsinaisessa käytössä olevana korkean tason ohjelmointikielenä. Fortranin lisäksi 50-luvun merkittäviin ohjelmointikieliin kuuluu kaupallisiin sovelluksiin suunniteltu COBOL. Kumpikin näistä ohjelmointikielten pioneereista on selvinnyt nykypäivään saakka, ja niitä päivitetään edelleen.

1950- ja 1960-lukujen vaihteessa kehitettiin ALGOL, joka on aikansa merkittävimpiä ohjelmointikieliä, sillä sen yhteydessä määriteltiin ohjelmointikielen formaaliin kuvaamiseen käytetty Backus-Naur-muoto (engl. *Backus Naur form, BNF*). 1960-luvulla ilmestyivät myös ensimmäinen luokat (engl. *classes*) sisältävä ohjelmointikieli, Simula, sekä C:n edeltäjäkielet.

Fortran

Tietokoneiden tullessa aiempaa edullisemmiksi ohjelmoijien palkkakulut alkoivat kuluttamaan budjetista aiempaa suuremman osan, ja ongelman suuruus vain jatkoi kasvamistaan tietokoneiden hintojen laskiessa. IBM:llä työskennellyt Fortranin kehittäjä John Backus toteaakin juuri taloudellisten kustannusten olleen yhtenä päämotivaattoreista, joka johti hänet ehdottamaan esihenkilölleen Fortranin kehittämistä. Backus kirjoittaa myös uskovansa, että Fortranin kaltaisen ohjelmointia helpottavan järjestelmän tarve oli syynä sille, että ylemmät tahot vastasivat projektin alati kasvaviin tarpeisiin tulevien vuosien aikana. [3]

Fortran on lyhenne nimestä Formula Translating System (toisinaan myös Formula Translator tai Formula Translation). Kieli suunniteltiin tieteelliseen käyttöön,

ja se onkin erityisen hyvä käsittelemään lukuja ja suorittamaan laskentaa [4]. Julkaisuvuonnaan 1957 Fortran mahdollisti ensimmäisenä ohjelmointikielenä symbolisten aritmeettisten lausekkeiden käytön [5]. Nykyohjelmointikielissä itsestäänselvyytenä pidetty ominaisuus mahdollisti lausekkeiden, kuten C = A / B, käytön, mikä tarkoitti sitä, että ohjelmoijat, jotka olivat 1950-luvulla pitkälti matemaatikoita ja muita tieteilijöitä, kykenivät kirjoittamaan ohjelmia itselleen jo entuudestaan tutulla syntaksilla [6].

Kielen ensimmäisen version, Fortran I:n, ominaisuuksiin kuuluivat muun muassa kokonaisluvut, liukuluvut, taulukot (engl. *array*), *DO-silmukka*, kolmisuuntainen *IF-lause*, *GO TO -lause* sekä joitain laitteistokohtaisia ominaisuuksia, sillä kieli oli suunnitteltu IBM 704 -tietokonetta varten. Alla on alkuperäisestä Fortran-käyttöoppaasta hieman karsittu koodiesimerkki suurimman arvon etsimisestä taulukosta [7]. Esimerkistä on jätetty pois arvojen lukeminen taulukkoon sekä tuloksen tulostaminen.

```
      DIMENSION A(999)


      BIGA = A(1)
      DO 20    I = 2, N
      IF (BIGA - A(I)) 10, 20, 20

10    BIGA = A(I)
20    CONTINUE
```

Kuva 2.1: Fortran-ohjelma suurimman arvon etsimiseen taulukosta.

Mielenkiintoisin ominaisuus yllä olevassa koodissa on kolmisuuntainen IF-lause, joka evaluoinnin lopputuloksen perusteella voi siirtyä kolmeen eri kohtaan ohjelman suoritusta sen mukaan, onko arvo alle, tasan vai yli nollan. Kolme lukua IF-lauseen lopulla (tässä tapauksessa 10, 20, 20) merkitsevät numeroituja rivejä, joille suoritus

hyppää BIGA - A(I) -laskutoimituksen arvon perusteella. Lause 20, joka sisältää CONTINUE-komennon, ei tee itsessään mitään, vaan sitä käytetään pääasiassa silmukoissa suorituksen hypyn kohteena. Ilman CONTINUE-komentoa — tai jotain muuta numeroitua riviä, johon suoritus voidaan ohjata — yllä olevassa toteutuksessa ei olisi mahdollista hypätä IF-lauseesta lauseen 10 yli.

Vuosien varrella Fortran on saanut lukuisia päivityksiä, jotka ovat tuoneet uusia ominaisuuksia sekä poistaneet vanhoja, kuten laitteistokohtaiset IBM 704 -tietokonetta varten olleet toiminnallisuudet. Fortranilla on edelleen uskollinen käyttäjäkuntansa; se on käytössä tieteellisissä sovelluksissa ja sitä on käytetty myös esimerkiksi Pythonin SciPy-kirjaston kehittämisessä [8].

COBOL

Vuonna 1955 Grace Hopper kollegoineen alkoi kehittää UNIVAC-tietokoneelle MATH-MATIC-ohjelmointikieltä, joka julkaistiin vuonna 1957. Ryhmä ei kuitenkaan erityisesti panostanut koodin konekielelle kääntämisen tehokkuuteen; tämän tuloksena käännetyt ohjelmat olivat suorituskyvyltään hitaita rajoittaen MATH-MATIC:in käyttökohteita. Samoihin aikoihin Hopperin tiimi kuitenkin loi kääntäjän ohjelmointikielelle, joka nimettiin ensin B-0:ksi, sitten Procedure Translatoriksi, ja lopulta nimeksi tuli FLOW-MATIC. MATH-MATIC:in tapaan FLOW-MATIC käytti englanninkielisiä avainsanoja, mutta aiempaa laajemmin, ja kieli keskittyi enemmän yritysten tarpeisiin. FLOW-MATIC vaikutti myöhemmin julkaistavan COBOL:in kehitykseen merkittävästi. [9]

Vuonna 1959 Pennsylvanian yliopistolla järjestettiin tapaaminen, jossa tietokoneiden parissa työskentelevät ihmiset (sekä akatemian että yritysten puolelta) keskustelivat mahdollisuudesta kehittää laitteistosta riippumaton ohjelmointikieli kaupallisia käyttökohteita varten. Yrityskäyttöön suunnitellun kielen nimeksi tulikin lopulta COBOL, joka on lyhenne sanoista "common business-oriented language". CO-

BOL:in suunnittelusta vastaavassa komiteassa oli erimielisyyksiä siitä, miten aritmeettisten operaatioiden ohjelmointi toteutettaisiin. Matematiikkaa osaavat henkilöt halusivat, että ohjelmoija voisi kirjoittaa matemaattisia kaavoja suoraan koodiin. Yritysmaailmaa edustavat taasen halusivat, että kielessä olisi avainsanat perusoperaatioille, kuten ADD, SUBTRACT, MULTIPLY ja DIVIDE. Komitea päätyi lopulta kompromissiin, jossa kielestä löytyisi edellä mainitut avainsanat, mutta niiden lisäksi myös avainsana COMPUTE, jonka kanssa matemaattisten kaavojen käyttö oli mahdollista. [10]

Alkuperäisen dokumentaation mukaan COBOL-ohjelma koostuu kolmesta osiosta: proseduuri-osiosta (engl. *procedure division*), data-osiosta sekä ympäristö-osiosta (engl. *environment division*). Proseduuri-osio sisältää ohjelman varsinaisen toiminnallisuuden, esimerkiksi laskutoimitukset. Data-osiossa kuvaillaan tiedostot, joita ohjelma lukee ja käsittelee, sekä ohjelman ajon aikana käytettävät muuttujat ja niiden tyypit. Tiedostoista luettavia tietoja varten on olemassa tietue-tyyppi (engl. *record*), jolla voidaan ryhmitellä toisiinsa liittyviä eri (tai samaa) tyyppiä olevia tietoja yhdeksi kokonaisuudeksi. Ympäristö-osiossa käsitellään laitteistokohtaiset yksityiskohdat, esimerkiksi käytettävissä olevan muistin määrä. [11]

Vuoteen 1961 mennessä kieleen lisättiin myös neljäs osio: tunniste-osio (engl. *identification division*). Tunniste-osiossa on lisätietoja ohjelmasta, muun muassa ohjelman nimi. COBOL saavutti suuren käyttäjäkunnan erityisesti finanssialalla, ja se on taustalla vielä nykyäänkin useissa pankkijärjestelmissä.

ALGOL

1950-luvun loppupuolella amerikkalaisista sekä eurooppalaisista tietojenkäsittelytieteilijöistä koostuva komitea suunnitteli ja kehitti ALGOL-ohjelmointikielen — tai tarkemmin sanottuna ohjelmointikieliperheen — joka saa nimensä sanoista ”Algo-

rithmic Language" [4]. Tavoitteena oli luoda olemassa olevia ehdotuksia yhdistelemällä standardi tieteellistä ohjelmointia varten [12].

ALGOL 60:n syntaksi määriteltiin nykyään nimellä Backus-Naur-muoto (engl. *Backus-Naur Form, BNF*) tunnetun notaation mukaan. BNF määrittelee, miten ohjelmointikielten syntaksi kuvaillaan, ja sitä käytetään edelleen ohjelmointikielten määrittelyssä. BNF:n lisäksi ALGOL 60:n mukana tuli muitakin innovaatioita: kielessä oli mahdollista määritellä lohkoja (engl. *block*), joilla oli mahdollista organisoida koodia sekä rajata muuttujien näkyvyyttä (engl. *scope*). ALGOL 60 tuki myös rekursiota sekä dynaamista muistinhallintaa. [5]

ALGOL:sta on kymmeniä eri versioita sekä laajennuksia, joita ei olennaisimpia lukuun ottamatta käsitellä tässä tutkielmassa. ALGOL on ollut myös inspiraationa ALGOL-perheen ulkopuolisille ohjelmointikielille, joita käsitellään tulevissa osioissa. Seuraava yksinkertainen koodiesimerkki on vuonna 1971 Carnegie Mellon -yliopistossa kirjoitetusta Algol 60 -oppaasta suurimman yhteisen tekijän laskemiseen [13].

```
r := 1;
        while r ≠ 0 do
        begin
                q := a ÷ b;
                r := a − q ∗ b;
                a := b;
                b := r
        end;
gcd := a
```

Kuva 2.2: ALGOL 60 -ohjelma suurimman yhteisen tekijän laskemiseen.

Huomionarvoista on, että ALGOL 60 -ohjelmakoodi muistuttaa jo monin tavoin rakenteeltaan nykyaikaisempia ohjelmointikieliä. Asetusoperaattori := vastaa mo-

nesta ohjelmointikielestä tuttua = -operaattoria. Begin ja end -avainsanat toimivat lohkojen avaajina ja sulkijoina, vastaten nykyohjelmointikielistä tuttuja kaarisulkeita {}. Joidenkin koodirivien lopussa on puolipiste, jolla erotellaan lauseet toisistaan. Yllä olevan koodin toimivuuteen on kriittistä huomioida, että ÷ suorittaa kokonaislukujaon. ALGOL 60 sisältää myös / -operaattorin tavallista, desimaalit säilyttävää jakolaskua varten.

Innovaatioistaan huolimatta ALGOL jäi lähinnä tutkijoiden käyttöön; yritykset eivät innostuneet kielestä, sillä se ei sisältänyt valmiita prosesseja siirräntään (engl. *input/output*) [4]. Syy siirrännän puuttumiselle oli kyseisten prosessien riippuvaisuus käytetystä laitteistosta, joka oli ristiriidassa sen kanssa, että ALGOL oli suunniteltu mahdollisimman riippumattomaksi käytetystä laitteistosta [14].

Näennäisesti vähäisistä käytännön käyttökohteistaan huolimatta ALGOL on toiminut inspiraationa monille myöhemmille ohjelmointikielille, joista osa on ALGOL:in eri versioita, ja osa kokonaan uusia ALGOL-perheen ulkopuolisia kieliä.

Simula

60-luvun alkupuolella luotu Simula (SIMUlation LAnguage) on Norjalaisten Ole-Johan Dahlin ja Kristen Nygaardin kehittämä ohjelmointikieli, jonka perustana on ALGOL 60. Tavoitteena oli luoda ohjelmointikieli helpottamaan simulaatioiden tekemistä, sillä tehtävä oli olemassa olevilla kielillä (Fortran, ALGOL) vaivalloinen. Kielen ensimmäinen versio, Simula I, sisältää ALGOL 60:n ominaisuudet, joiden lisäksi siihen on lisätty prosessit (engl. *process*), jotka mahdollistavat ”rinnakkaisen” ajon. Prosessit muistavat suorituksensa tilan, joka mahdollistaa eri prosessien välillä edestakaisen siirtymisen suorituksen tilan säilyessä ennallaan. [15]

Vuonna 1967 julkaistiin versio nimellä Simula 67, joka toi mukanaan uusia ominaisuuksia. Olennaisimpana lisäyksenä oli luokat (engl. *class*) sekä niistä muodostettavat oliot (engl. *object*). Myös aliluokkien (engl. *subclass*) luominen sekä perintä

(engl. *inheritance*) oli mahdollista. Uusina muuttujien tyyppeinä (engl. *type*) kieleen lisättiin ”character” sekä ”text” helpottamaan merkkijonojen käsittelyä. Uusien ominaisuuksien myötä Simula 67 siirtyi Simula I:n simulaatiopainotteisesta ohjelmointikielestä lähemmäs yleiskäyttöistä ohjelmointikieltä. [16]

Simula sisältää automaattisen roskienkeruun (engl. *garbage collection*) pitämällä lukua viittausten määristä [17]. Roskienkeruun tarkoituksena on vapauttaa muistialueet uudelleenkäytettäväksi, kun niihin ei ole enää viittauksia. Simula ei saavuttanut suurta käyttäjäkuntaa, mutta erityisesti sen olio-ohjelmointiin liittyvät ominaisuudet vaikuttivat merkittävästi myöhempien kielten kehitykseen.

Pascal

ALGOL:in toiminnallisuuksia laajentamaan päätyi myös Niklaus Wirth, joka oli mukana ALGOL-työryhmässä 1960-luvulla. Uusilla tietotyypeillä ja muilla pienillä lisäyksillä paranneltu kieli sai nimekseen ALGOL W. Saadessaan professuurin vuonna 1968 teknillisessä korkeakoulussa Sveitsissä, Wirth koki, ettei olemassa olevat ohjelmointikielet olleet optimaalisia ohjelmoinnin opettamista varten. [18]

Tyytymättömyys olemassa olevien ohjelmointikielten rakenteiden epäloogisuuteen ajoi Wirthin kehittämään Pascal-ohjelmointikielen. Perustana Pascalille oli ALGOL 60, josta Wirth otti kieleensä ominaisuudet, jotka eivät olleet kielen luonnollisen kuvaamisen esteenä. Wirth pyrki luomaan ohjelmointikielen, jolla voitaisiin luoda tehokasta sekä luotettavaa koodia, ja joka olisi luonteva kieli ohjelmoinnin opettamiseen [19]. Yritysten, hallituksen ja teollisuuden tuen puutteesta huolimatta Pascal sai laajan käyttäjäkunnan, joka levitti tietoa kielestä. Pascal oli pitkään käytössä erityisesti ohjelmoinnin opetuksessa [18].

CPL, BCPL ja B

1960-luvulla Cambridgen ja Lontoon yliopistojen yhdessä kehittämä CPL (Combined Programming Languge) on jälleen yksi ALGOL 60:n pohjalta luoduista ohjelmointikielistä. Se tarjosi muun muassa kolmiosaisen ehto-operaattorin (engl. *ternary conditional operator*), joka toimi nykyohjelmointikielistäkin tutulla tavalla:

```
ehto → lauseke1 , lauseke2
```

Kuva 2.3: CPL-ohjelmointikielen kolmiosaisen ehto-operaattorin muoto.

Ehdon ollessa tosi, suoritetaan lauseke1, muussa tapauksessa suoritetaan lauseke2. Myös lausekkeet voivat sisältää kolmiosaisen ehto-operaattorin, jolloin rakenne vastaa sisäkkäisiä if-else-lauseita. CPL:ssä koodilohkot eroteltiin avainsanojen begin ja end sijaan käyttämällä pykälämerkkiä § (lohkon sulkeva pykälämerkki sisälsi pystyviivan symbolin päällä). [20] CPL jäi verrattain vähän käytetyksi ohjelmointikieleksi, mutta se on kuitenkin olennainen askel ALGOL 60:n ja C-kielen välillä.

Cambridgen yliopistossa työskennellyt Martin Richards jatkokehitti CPL:stä uuden ohjelmointikielen nimeltä BCPL (Basic Combined Programming Language). Kuten nimikin antaa ymmärtää, BCPL oli kevennetty versio CPL:stä; se karsi ominaisuuksia CPL:stä, jotta kieli toimisi tehokkaammin järjestelmäohjelmoinnissa (engl. *systems programming*). Poikkeuksellisena ratkaisuna BCPL sisälsi vain yhden tyypin (engl. *type*), bittijonon (engl. *binary bit pattern*). Arvojen tulkinta oli riippuvainen käytetystä operaattorista; esimerkiksi operaattorit +, -, / ja * käsittelivät arvoja kokonaislukuina. [21]

Dennis Ritchie ja Ken Thompson kehittivät BCPL:n pohjalta B-ohjelmointikielen. B muutti asetusoperaattorin Fortranin tapaan yksittäiseksi yhtäsuuruusmerkiksi monesta muusta kielestä tutusta := -syntaksista poiketen. B esitteli myös lisäys- ja vähennysoperaattorit (engl. *increment and decrement operators*) ++ sekä --, jotka lisäsivät tai vähensivät operandin arvoa yhdellä, joko ennen tai jälkeen operandin

arvon tulkitsemista, riippuen oliko operaattori ennen vai jälkeen operandin (++arvo tai arvo++).

2.2 1970-luku

1970-luvulla kehitettiin C-ohjelmointikieli, joka tunnetaan suorituskyvystään, mutta myös vaikeudestaan ohjelmointikielenä sen mahdollistamien ohjelman vakauden vaarantavien ohjelmointivirheiden vuoksi. C:n lisäksi alkunsa 1970-luvulla sai Smalltalk, joka on ensimmäinen olio-ohjelmointiparadigmaa noudattava ohjelmointikieli. Myös erittäin pitkän suunnittelu- ja kehitystyön takana ollut Ada sai alkunsa 1970-luvulla, vaikka varsinainen julkaisu kestikin 1980-luvulle saakka.

C

B-ohjelmakoodin kääntämisen hitaus sekä BCPL:stä peritty tyyppien puuttuminen johti Ritchien jatkokehittämään B:stä uuden ohjelmointikielen, jota hän kutsui nimellä NB (new B). Olennaisimpana muutoksena Ritchie lisäsi uuteen kieleen tyypityksen. Myöhemmin, halutessaan tehdä pesäeroa B-kieleen, Ritchie uudelleennimesi luomansa NB-kielen C:ksi. [22]

C on suorituskyvyltään tehokas ohjelmointikieli, joka mahdollistaa — tai jopa rohkaisee — käyttäjänsä kirjoittamaan vakaudeltaan sekä tietoturvaltaan kyseenalaisia ohjelmia. Erityisen vaarallisia virheitä C ohjelmissa aiheuttaa osoittimien (engl. *pointer*) käsittelyssä tapahtuvat ohjelmointivirheet, jotka usein johtavat puskuriylivuotoon (engl. *buffer overflow*), joka tarkoittaa yksinkertaistettuna jonkin arvon tallentamista kokonaan tai osittain sille varatun muistialueen ulkopuolelle. Virhealttiuden kääntöpuolena C mahdollistaa turvallisempia kieliä matalammalla tasolla (lähempänä laitteistoa) olevan hallinnan, jota pidetään erityisen tärkeänä järjestelmäohjelmoinnissa. [23]

C on selvinnyt nykypäivään saakka erittäin laajasti käytössä olevana ohjelmointikielenä. Suorituskykynsä ansiosta sitä käytetään muun muassa käyttöjärjestelmissä, sulautetuissa järjestelmissä ja kääntäjissä. Suoran C-ohjelmoinnin lisäksi ohjelmoija saattaa tietämättään käyttää muissa ohjelmointikielissä moduuleja, jotka ovat tehty suorituskyvyn vuoksi C:llä.

Smalltalk

1970-luvulla alkunsa sai erittäin merkittävä ohjelmointiparadigma: olio-ohjelmointi. Vaikka Simula oli tuonut ohjelmoijien työkalupakkiin luokat jo aikaisemmin, varsinainen olio-ohjelmointi konseptina sai alkunsa kuitenkin vasta Smalltalkin myötä 1970-luvulla. Smalltalk kehitettiin Xeroxin PARC-tutkimuskeskuksessa pääasiassa Alan Kayn, Dan Ingallsin sekä Adele Goldbergin toimesta. Kayn kokemukset Simulan sekä Lispin kanssa vaikuttivat hänen perspektiiviinsä ohjelmoinnin suhteen ja täten Smalltalkin ominaisuuksiin. Keskeisimpiä ajatuksia Smalltalkin kehityksessä oli, että kaikki ohjelman osat koostuvat olioista, jotka ovat aina jonkin luokan esiintymiä. Tämä pätee myös primitiivityyppeihin (kokonaisluvut, booleanit ym.) Oliot pitävät itse huolta omasta tilastaan, ja niillä on edustamaansa luokkaan kuuluvat metodit, joilla tilaa voidaan muuttaa. Oliot voivat kommunikoida keskenään lähettämällä ”viestejä”. Viestit sisältävät kutsuttavan olion nimen, kutsuttavan metodin nimen sekä mahdollisen argumentin. Viestin vastaanottanut olio tulkitsee viestin, tekee mahdolliset toimenpiteet ja vastaa lähettäjälle. [24]

Smalltalkista inspiroituneet David Ungar sekä Randall Smith suunnittelivat Self-ohjelmointikielen, joka tuli käyttöön vuonna 1987. Smalltalkin tapaan Selfissä ohjelma muodostuu kokonaan olioista, mutta Smalltalkista poiketen Selfissä ei ole luokkia. Luokkien sijaan oliot ovat prototyyppejä (engl. *prototype*), joita voidaan vapaasti kloonata. Tyypillisesti olio-ohjelmoinnissa jokainen erilainen olio vaatii oman

luokkansa, mutta Selfissä erikoistapaukset voidaan hoitaa kloonaamalla olemassa oleva olio ja lisäämällä (tai poistamalla) sille tarvittavat ominaisuudet. [25]

Ada

Vuonna 1975 Yhdysvaltain puolustusministeriö (engl. *United States Department of Defense*) perusti työryhmän uuden ohjelmointikielen määrittämistä varten. Ohjelmistokehityksen kulut olivat alkaneet kuluttamaan alati suurempaa osaa resursseista. Puolustusjärjestelmien koostuessa useista itsenäisistä, keskenään kommunikoivista laitteista, puolustusministeriössä katsottiin, että ohjelmointikielen standardoinnilla voitaisiin pienentää ohjelmistokehityksestä koituvia kuluja. Työryhmä loi uuden ohjelmointikielen vaatimuksista dokumentin, joka lähetettiin puolustusorganisaation eri osastoille, akateemiselle yhteisölle, valtion virastoille ja muille kiinnostuneille kommentointia varten. [26]

Puolustusministeriö järjesti suunnitteluprosessista kilpailutuksen, johon osallistui 20 eri tahoa. Yksi osallistujista oli ranskalaisen Jean Ichbiahin tiimi, joka lopulta valikoitui voittajaksi vuonna 1979 [27]. Kieli sai nimekseen Ada, 1800-luvulla eläneen matemaatikon ja Lovelacen kreivittären Augusta Ada Byronin mukaan, joka tunnetaan työstään Charles Babbagen keskeneräiseksi jääneen projektin, analyyttisen koneen (engl. *analytical engine*), parissa työskentelystä. Ichbiahin johdolla kielelle luotiin standardi, jonka yhdysvaltalainen standardeja valvova ANSI (American National Standards Institute) hyväksyi vuonna 1983. Ensimmäiset kääntäjät Adalle valmistuivat vuonna 1983 [27].

Ada otti vaikutteita muun muassa Pascalista, Algol 68:sta sekä Simulasta. Ada mahdollistaa prosessien rinnakkaisen suorittamisen tehtävien (engl. *task*) avulla. Tehtävät ovat verrattavissa esimerkiksi Javan säikeisiin (engl. *thread*). Kielessä on lisäksi kattava ajonaikaisten poikkeusten käsittely. Koodin uudelleenkäytettävyyttä

lisää pakkaukset (engl. *package*), jotka mahdollistavat itsenäisten ominaisuuksien erottamisen omaan moduuliinsa, ja niiden kutsumisen muista pakkauksista. [28]

2.3 1980-luku

1980-luvulla olio-ohjelmointi otti lisää askeleita eteenpäin, kun Bjarne Stroustrup kehitti C++:n, joka yhdisti C:n tehokkuuden olio-ohjelmointiparadigman kanssa. 1980-luvulla alkunsa sai myös Pythonin edeltäjä, ABC, joka kehitettiin Alankomaissa toimivassa tutkimuskeskusessa.

C++

Bell Labsilla työskenellyt Bjarne Stroustrup kehitti vuonna 1979 esikääntäjän (engl. *preprocessor*), joka käänsi Stroustrupin vastikään luoman C with Classes -ohjelmointikielen koodin C-koodiksi. C with Classes oli C:n laajennus, johon Stroustrup oli lisännyt Simulan inspiroimat luokat, tyyppitarkastukset (engl. *type checking*) ja muitakin lisäyksiä. Stroustrupille tärkeätä kielen kehityksessä oli, että sen suorituskyky pysyisi C:n veroisena, jotta sillä voitaisiin ohjelmoida ratkaisuja mihin tahansa ongelmiin mitä C:llä voitiin ratkaista, mutta selkeämmällä rakenteella. [29]

1980-luvun alkupuoliskolla Stroustrupin tavoitteet kielen suhteen kasvoivat; se nimettiin C84:ksi ja Stroustrup alkoi kehittää sille kääntäjää, jonka nimi oli Cfront. Hieman poikkeuksellisesti Cfront käänsi koodin edelleen C-koodiksi, mutta koodin syntaksin ja semantiikan tarkistaminen oli Cfrontin tehtävä; C:n kääntäjää käytettiin vain koodin generoimiseen. [29]

Lopulta kieli sai nimen C++ ja sille julkaistiin vuonna 1985 opas ”The C++ Programming Language”. Seuraavat vuodet C++:n käyttäjäkunta kasvoi räjähdys-

mäisesti. C++ on saanut (ja saa edelleen) lukuisia päivityksiä ja se on nykyäänkin yksi suosituimmista ohjelmointikielistä.

ABC

Alankomaissa Centrum Wiskunde & Informatica (CWI) -tutkimuskeskuksessa 1980-luvulla Leo Geurtsin, Lambert Meertensin sekä Steven Pembertonin kehittämä ABC-ohjelmointikieli tarjoaa tekijöidensä mukaan helposti luettavan kielen, jonka korkean tason operaatiot mahdollistavat muita aikansa ohjelmointikieliä lyhyemmät ohjelmat; yksittäinen operaatio voi vastata lukuisia rivejä C tai Pascal -koodia [30]. Kääntöpuolena on ohjelmien suorituksen hitaus muihin kieliin verrattuna. ABC poikkeaa aiemmin käsitellyistä ohjelmointikielistä lisäksi koodin kääntämisessä; ABC käyttää kääntäjän sijaan tulkkia, joka kääntää koodia vasta ajon aikana. ABC-projektissa mukana ollut Guido van Rossum mainitsee ABC:n kehitykseen vaikuttaneiksi kieliksi ALGOL 68:n [31] sekä SETL:n [32], joka sekin on johdettu ALGOL:sta [33].

Alla olevassa koodiesimerkissä on pienimuotoinen ”puhelinluettelosovellus” ABC:lla koodattuna [30].

```
PUT {} IN tel
PUT 4133 IN tel["Leo"]
PUT 4141 IN tel["Doug"]
PUT 4166 IN tel["Paul]

FOR name IN keys tel:
        WRITE name, ":", tel[name] /
```

Kuva 2.4: ABC-ohjelmointikielellä tehty pienimuotoinen puhelinluettelosovellus.

Begin ja end -avainsanojen tai kaarisulkeiden sijaan ABC:n koodilohkot erotetaan toisistaan sisennyksen mukaan. Yksinäinen kauttaviiva WRITE-komennon

(tulostuskomento) lopussa lisää tulosteeseen rivinvaihdon. Pythonia tunteva lukija ei välttämättä ylläty yllä olevaa koodia lukiessaan, että aiemmin mainittu Guido van Rossum on Pythonin luoja, sillä yhtäläisyyksiä ABC:n ja Pythonin välillä on havaittavissa.

2.4 1990- ja 2000-luku

1990-luvulla sai alkunsa Python, jonka tarkoitus oli toimia nopeana skriptikielenä pieniin ongelmiin, joiden ratkaiseminen tavanomaisilla ohjelmointikielillä olisi tarpeettoman vaivalloista. 90-luvun alkupuolella maailmanlaajuinen verkko (engl. *World Wide Web*) mullisti tiedonjakamisen, kun internet alkoi täyttyä verkkosivuista. Verkkosivut kuitenkin sisälsivät lähinnä staattista tekstiä ja kuvia ilman monimutkaisempia toiminallisuuksia. Verkkosivuja päästiinkin pian elävöittämään JavaScriptillä sekä PHP:lla, ja osin myös Javalla, jonka myöhemmät käyttökohteet kuitenkin ovat huomattavasti sen ensimmäisiä käyttökohteita laajempia. 1990-luvulla on myös C#:n juuret; Microsoftin kehittämä olio-ohjelmointikieli julkaistiin nopeahkon kehitystyön jälkeen vuonna 2000.

Python

Työskennellessään CWI:llä 1980-luvun lopulla Guido van Rossum sai tehtäväkseen luoda uuden skriptikielen tutkimuskeskuksen tarpeisiin. Van Rossum otti kieleen hyviksi katsomansa ominaisudet ABC:stä ja lisäsi tai muutti muita ominaisuuksia oman makunsa mukaiseksi. Vuonna 1991 julkaistun skriptikielen nimeksi tuli Python. Yhtenä merkittävimmistä päätöksistä kielen kehityksen suhteen van Rossum pitää sen laajennettavuutta; siihen oli mahdollista luoda uusia moduuleja sekä Pythonilla että C:llä. Moduuleilla oli mahdollista lisätä uusia tyyppejä ja muita omi-

naisuuksia kieleen. Lopulta kielestä tuli jotain skriptikielen ja korkean tason ohjelmointikielen väliltä, tai niiden ominaisuuksia yhdistävä kieli. [34]

Van Rossumin tavoite Pythonille oli, että se olisi toissijainen ohjelmointikieli C ja C++ -ohjelmoijille tilanteisiin, joissa C-ohjelman kirjoittaminen olisi tarpeettoman vaivalloista käyttökohteeseen nähden. Suunnitellusta koodin ”kertakäyttöisyydestä” huolimatta Python on päätynyt käyttöön myös ensisijaisena ohjelmointikielenä pysyviin ratkaisuihin. [35] Python on lisäksi laajasti käytössä tekoäly- ja koneoppimisprojekteissa.

Java

1990-luvulla alkunsa on saanut myös internetin varaan rakennettu World Wide Web (WWW), joka mahdollisti tiedon jakamisen internetin yli muun muassa verkkosivujen muodossa. Ensi alkuun pelkällä HTML:llä tehdyt verkkosivut loivat kysynnän työkaluille, joilla verkkosivuista voitaisiin saada monipuolisempia ja interaktiivisempia.

Sun Microsystemsillä työskennellyt James Gosling suunnitteli alun perin kuluttajaelektroniikan sulautettuja järjestelmiä varten Oak-ohjelmointikielen. Kieltä jatkokehitettiin vuosia lukuisten henkilöiden toimesta ja se uudelleennimettiin Javaksi, joka julkaistiin vuonna 1995. Java on perinyt ominaisuuksia C:stä ja C++:sta, pyrkien kuitenkin vähentämään C-kielille tyypillisiä ohjelmointivirheitä, tai ainakin saamaan virheet esiin jo kääntämisen yhteydessä. Java on vahvasti tyypitetty, automaattisen roskienkeruun sisältävä olio-ohjelmointikieli, joka käännetään tavukoodiksi, jota ajetaan Java-virtuaalikoneella (engl. *Java virtual machine, JVM*). [36]

Javan alkuaikoina kieltä käytettiin enimmäkseen sovelmien (engl. *applet*) luomiseen. Sovelmat ovat web-selaimessa suoritettavia pieniä sovelluksia. Sovelmat mahdollistivat staattisten HTML-sivujen muuttamisen dynaamisiksi liikkuvalla tekstillä,

grafiikalla sekä äänitehosteilla [37]. Nykyään Java kuuluu suosituimpien ohjelmointikielten joukkoon ja sitä käytetään sovelmien sijaan monenlaiseen ohjelmistokehitykseen.

JavaScript

Java ei suinkaan ollut ainoa kieli, jolla staattisia verkkosivuja pyrittiin elävöittämään. Netscapella työskennellyt Brendan Eich kehitti vuonna 1995 skriptikielen, jota suoritettaisiin käyttäjien tietokoneilla verkkoselaimen toimesta [38]. Kieli oli alun perin nimeltään LiveScript (samalla nimellä on olemassa myös 2011 julkaistu ohjelmointikieli), mutta nimettiin myöhemmin JavaScriptiksi. Nimestään huolimatta JavaScript ei suoranaisesti liity hieman aikaisemmin samana vuonna julkaistuun Javaan. Blogissaan Eich kertoo JavaScriptin ottaneen vaikutteita Javasta, Schemestä sekä Selfistä [39]. Ensimmäinen JavaScriptiä tukenut selain oli Netscapen kehittämä Netscape Navigator. Navigatorin kilpailijaselain, Microsoftin kehittämä Internet Explorer, tuki JavaScriptin toteutusta, jonka Microsoft nimesi JScriptiksi [40].

JavaScriptiä — tai sen käyttäjiä — on historiallisesti vaivannut toisistaan poikkeavat toteutukset eri selaimissa sekä yksittäisenkin selaimen eri versioiden välillä [41]. Kehittäjät eivät voi tietää millä selaimella käyttäjät tulevat verkkosivuille, mutta myöskään koodin toimivuutta kaikilla mahdollisilla selaimilla ja selainversioilla ei ole realistista varmistaa.

Vuonna 1997 JavaScriptille luotiin standardi nimeltä ECMAScript, joka ohjaa JavaScript-toteutusten kehitystä [42]. Vuosien varrella JavaScript on Pythonin tapaan hämärtänyt väliä skriptikielen ja kokonaisvaltaisen ohjelmointikielen välillä sekä päätynyt käyttäjien selainten lisäksi myös palvelinten puolelle käyttöön. Nykypäivänä on epätodennäköistä päätyä verkkosivulle, jossa ei ole JavaScriptiä käytössä.

PHP

Javan ja JavaScriptin lisäksi web-kehitykseen julkaistiin vuonna 1995 kolmaskin kieli; Rasmus Lerdorfin kehittämä PHP on skriptikieli verkkosivujen kehittämiseen. JavaScriptin tavoin PHP on käytössä web-kehityksessä ”laajentamassa” HTML:n toiminnallisuutta. PHP on ottanut vaikutteita C:stä, Javasta sekä Perlistä [43]. Alun perin nimi oli lyhenne sanoista Personal Home Page Tools, myöhemmin siitä tuli PHP: Hypertext Preprocessor. Kielen kasvupotentiaalia rajoitti se, että Lerdorf kehitti kieltä pääasiassa yksin ensimmäiset vuodet. Vuosien mittaan projektiin on kuitenkin tullut lisää kehittäjiä. [44]

C#

Vuosituhannen vaihteessa Microsoft kehitti C# (C Sharp) -ohjelmointikielen, joka on yhdistelmä C++:aa sekä Javaa [45]. C#:n ensimmäisen version loivat Anders Hejlsberg, Scott Wiltamuth ja Peter Golde, ja se julkaistiin vuonna 2000. C# standardoitiin ECMA Internationalin toimesta vuonna 2002.

ECMA:n spesifikaation mukaan C# on tarkoitettu kaikenkokoisten järjestelmien ohjelmointiin, pyrkimyksenään olla tehokas ja säästeliäs resurssien suhteen, muttei kuitenkaan kilpailla C:n kanssa suorituskyvyssä. C# on olio-ohjelmointikieli kuten C++, mutta se pyrkii tarjoamaan työkalut, joilla ohjelmoija voi luoda helpommin ja nopeammin ”turvallisia” ohjelmia. Yksi olennainen ominaisuus on automaattinen roskienkeruu, jonka ansiosta ohjelmoijan ei tarvitse allokoida ja deallokoida muistia manuaalisesti. C#:n kääntäjä pyrkii myös havaitsemaan tilanteita, joissa koodissa yritetään käyttää muuttujia, joille ei ole vielä asetettu arvoa (engl. *initialize*). [46]

Tutkielmassa on nyt käsitelty imperatiivisia korkean tason ohjelmointikieliä ja niiden historiaa 1950-luvulta vuosituhannen vaihteeseen saakka. Kieliä olisi vielä runsaasti enemmänkin, mutta ainakin olennaisimmat askeleet imperatiivisen ohjel-

moinnin kehityskulussa pitäisi olla käsiteltynä. Seuraavaksi käsitellyistä ohjelmointikielistä muodostetaan sukupuu.

3 Sukupuun muodostaminen

Ohjelmointikielten sukupuun muodostaminen ei ole täysin yksiselitteinen tehtävä. Erilaisia ohjelmointikieliä on satoja tai jopa tuhansia, riippuen laskentatavasta. Yksittäisellä ohjelmointikielellä voi olla kymmeniä eri toteutuksia (ALGOL), ja katsottaessa ohjelmointikielten ominaisuuksia ja niiden alkulähteitä havaitaan, että yksittäisestäkin ohjelmointikielestä on useita eri versioita, sillä kaikkia (käytössä olevia) kieliä päivitetään säännöllisesti tarpeen mukaan. Tällöin kattavassa sukupuussa pitäisi mahdollisesti ottaa huomioon mistä ohjelmointikielestä sekä mistä versiosta mikäkin ominaisuus on peritty. Lisäksi teoriassa kuka tahansa voi itse luoda ohjelmointikielen ja sille kääntäjän tai tulkin, joten täysin tuntemattomia ja dokumentoimattomia kieliä lienee lukuisia.

Runsaslukuisuuden lisäksi toinen sukupuun muodostamista hankaloittava asia on, että kaikki perintä ei tapahdu aina vain yhteen suuntaan, vaan esimerkiksi C# on ottanut mallia Javasta ja myöhemmin Java on ottanut mallia C#:sta [47].

Kaiken huomioon ottavan sukupuun muodostaminen olisi siis äärimmäisen työläs prosessi, ja lopputulos näyttäisi todennäköisesti lähinnä hämähäkinseitiltä lukuisten edestakaisin ohjelmointikielten välillä kulkevien yhteyksien vuoksi. Yllä olevista syistä tässä tutkielmassa sukupuuta muodostetaan tietyin rajauksin:

- Sukupuuhun tulee vain imperatiivisia ohjelmointikieliä.
- Imperatiivisista ohjelmointikielistä mukaan otetaan (subjektiivisesti) olennaisimmat kielet.

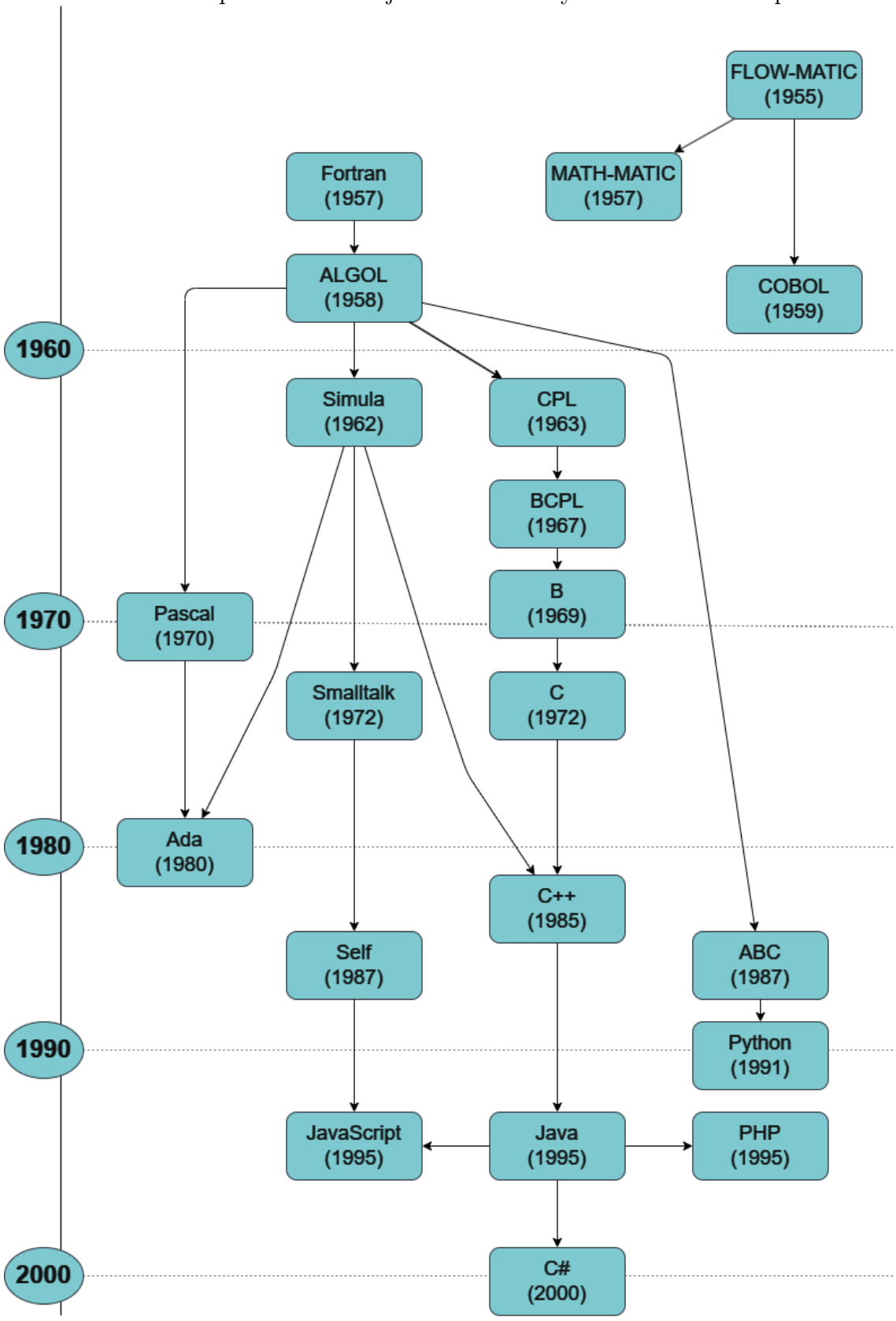
Olennaisiksi kieliksi katsotaan luvussa 2 esitellyt ohjelmointikielet. Kielet on valikoitu sen mukaan, että ne ovat joko ensimmäisiä korkean tason ohjelmointikieliä, ne ovat tuoneet ensimmäisinä merkittäviä ominaisuuksia, joita myöhemmät ohjelmointikielet ovat perineet, ne ovat ”välietappina” jollekin merkittävämmälle kielelle, tai ne ovat muutoin vaikuttaneet merkittävästi ohjelmointikielten kehitykseen. Tutkielmassa käytetyt yhteydet kielten välillä perustuvat luvussa 2 käytettyihin lähteisiin. Kaikkia eri vaikuttimia ei kuitenkaan löydetyissä lähteissä ole välttämättä mainittu, ja toisaalta kaikki kielet eivät mahtuisi tutkielmaan muutenkaan, joten on hyvä pitää mielessä muodostetun sukupuun olevan vain yksi mahdollinen tapa muodostaa se.

Tarkoituksena kuitenkin on, että sukupuusta löytyisi erityisesti nykyhetkenä tunnetuimmat ja käytetyimmät ohjelmointikielet sekä niille jokin järkevä polku ensimmäisiin korkean tason ohjelmointikieliin saakka. Lukija pystyy siis saamaan sukupuusta nopealla katsauksella pintapuolisen käsityksen imperatiivisten ohjelmointikielten keskinäisistä suhteista sekä niiden kronologisesta järjestyksestä julkaisuvuoden mukaan.

Sukupuun selkeyttämiseksi kielten väliset yhteydet on yksinkertaistettu siten, että suoraan saman haaran alla olevista kielistä ei piirretä yhteyttä mahdollisiin isovanhempiin. Toisin sanottuna, vaikka esimerkiksi Javaan vaikuttaneisiin kieliin lukeutuu sekä C että C++, on yhteys Javasta vain C++:aan, koska C++:sta yhteys jatkuu kohti C:tä.

Muodostetusta sukupuusta (sekä luvun 2 tekstistä) on havaittavissa, että ALGOL-ohjelmointikieliperhe lukuisine eri toteutuksineen on kieli, jota voidaan pitää erityisen merkittävänä osana imperatiivisten ohjelmointikielten historiaa, sillä lähes jokainen niistä pohjautuu tavalla tai toisella ALGOL:in ominaisuuksiin. Sukupuu löytyy seuraavalta sivulta.

Kuva 3.1: Imperatiivisten ohjelmointikielten yksinkertainen sukupuu.



4 Yhteenveto

Tutkielman tarkoituksena oli selvittää mikä on ohjannut imperatiivisten ohjelmointikielten kehitystä sekä mitkä imperatiiviset ohjelmointikielet ovat sukua toisilleen. Lisäksi tavoitteena oli muodostaa niille yksinkertainen sukupuu.

Ensimmäinen tutkimuskysymys oli: "Mitkä asiat ovat ohjanneet imperatiivisten ohjelmointikielten kehitystä?"

Tutkielmassa esiintyvien ohjelmointikielten kehitykseen vaikuttavia tekijöitä oli useita. Tietokoneiden ollessa vielä kalliita ja harvinaisia korkean tason ohjelmointikielten alkuaikoina, on merkittävää kehitystyötä tehty lähinnä suurissa instituutioissa, joilla on ylipäätään ollut varaa tietokoneisiin; joukkoon lukeutuu muun muassa yliopistoja, teknillisiä korkeakouluja, tutkimuskeskuksia, sotilasorganisaatioita sekä tietokoneita valmistavia yrityksiä. Myöhemmin tietokoneiden yleistyessä ohjelmointi on ollut suuremman yleisön tavoitettavissa, mikä on mahdollistanut kielten kehittämisen myös yksittäisille henkilöille, ainakin kehityksen alkuvaiheissa. Kehittäjät ovat kuitenkin tyypillisesti olleet tietojenkäsittelyn alalla töissä yliopistoissa, tutkimuskeskuksissa tai alan yrityksissä.

Kukin taho on kehittänyt ohjelmointikieliä kulloinkin vaadittavan käyttökohteen mukaan. John Backus loi Fortranin helpottamaan ohjelmointia, koska muita korkean tason ohjelmointikieliä ei vielä ollut käytössä. Grace Hopper tiimeineen kehitti COBOL:in kaupallisia käyttökohteita varten, kohderyhmänään ohjelmoijat, jotka eivät tunne matemaattista notaatiota. ALGOL sai alkunsa, koska ryhmä tietojenkäsit-

telytieteilijöitä halusi luoda standardoidun ohjelmointikielen. Myöhempiä kieliä on luotu alun perin rajatumpiin käyttökohteisiin: Simula simulaatioihin, Pascal ohjelmoinnin opettamiseen, C järjestelmäohjelmointiin. Olio-ohjelmoinnin ilmestyessä ja sen suosion kasvaessa käytetyimmät kielet olivat sitä tukevia: C++, Java, C# ym. Myös internetillä — erityisesti verkkosivuilla — on ollut osansa ohjelmointikielten kehityksessä; staattisten HTML-sivujen monipuolistaminen interaktiivisemmiksi kokonaisuuksiksi on ajanut JavaScriptin, PHP:n ja aivan Javan alkuvaiheissa myös sen kehitystä. Usein myös ohjelmointikielen kehittäjän aikaisemmat kokemukset ja mahdolliset turhautumiset muiden ohjelmointikielten käytössä on johtanut jonkin kielen ”kopioimiseen” ja siitä (kehittäjän mielestä) parannellun version luomiseen.

Uusien kielten kehittämistä motivoi lisäksi se, että vaikka Turing-täydellisellä ohjelmointikielellä on mahdollista ohjelmoida sama toiminnallisuus kuin millä tahansa muullakin Turing-täydellisellä kielellä, eroja löytyy kuitenkin sekä ohjelman suoritustehokkuudessa ja ohjelmoinnin vaikeudessa. Tietyt kielet siis sopivat paremmin tiettyihin tehtäviin. Onkin mielenkiintoista seurata kuinka kauan uusia ohjelmointikieliä vielä luodaan ja onko uusimpien kielten mahdollista saavuttaa yhtä laajaa käyttäjäkuntaa kuin esimerkiksi Javalla ja Pythonilla on. Jos on, mikä on se ominaisuus tai ero, joka saa suuren käyttäjäkunnan vaihtamaan tutuista ohjelmointikielistään kokonaan uuden kielen pariin.

Toinen tutkimuskysymys oli: "Mitkä imperatiiviset ohjelmointikielet voidaan tulkita toistensa sukulaisiksi?"

Kysymykseen ei ole täysin yksiselitteistä vastausta. Korkean tason ohjelmointikielten alkuvuosina erottelu olisi ollut selkeämpi, sillä eri kieliä oli huomattavasti vähemmän, ja ominaisuuksia ei ollut levinnyt edestakaisin kielten välillä. Voidaan kuitenkin tulkita, että ALGOL on mainittuna inspiraation lähteenä niin monen myöhemmän ohjelmointikielen kehityksessä, että käytännössä kaikki laajasti nyky-

ään käytössä olevat imperatiiviset ohjelmointikielet ovat sen jälkeläisiä tavalla tai toisella.

Luvussa kolme muodostettu sukupuu antaa kohtalaisen käsityksen kielten sukulaissuhteista, mutta vain yhdensuuntaisena alaspäin valuvana perintänä. Lisäksi se käsittelee ohjelmointikieliä yhtenä kokonaisuutena huolimatta siitä, että esimerkiksi Fortranin vuoden 1957 versio eroaa varmasti monella tavalla vuoden 2023 julkaisusta. Sukupuussa olisi mahdollista ottaa huomioon ohjelmointikielten eri versiot sekä eri kielten välillä tapahtunut edestakainen ominaisuuksien levittyminen, mutta sukupuun luettavuus kärsisi yhteyksien suuresta määrästä johtuen. Periaatteessa siis kaikki imperatiiviset ohjelmointikielet voitaisiin katsoa toistensa sukulaisiksi.

Jatkotutkimuksen osalta voisi olla mielenkiintoista pyrkiä kokoamaan kattavampi sukupuu, johon otetaan mukaan kieliä laajemminkin sekä muista paradigmoista että myös imperatiivisten kielten osalta niitä, jotka eivät tähän tutkielmaan päässeet. Aikaisemmin mainituista syistä tosin projektin koko kasvaa räjähdysmäisesti mitä enemmän kieliä mukaan otetaan, sillä yksittäisenkin kiinnostavan kielen lisääminen saattaa johtaa siihen, että mukaan pitää ottaa myös sen edeltäjiä, joita työssä ei välttämättä vielä ole mainittu.

Lähdeluettelo

[1] L. S. Vailshery. "Most used programming languages among developers worldwide as of 2024". (2024), url: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ (viitattu 16. 08. 2024).

[2] A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society*, vol. s2-42, nro 1, s. 230–265, 1937. DOI: https://doi.org/10.1112/plms/s2-42.1.230.

[3] J. Backus, "The history of Fortran I, II, and III", *IEEE Annals of the History of Computing*, vol. 20, nro 4, s. 68–78, 1998. DOI: 10.1109/85.728232.

[4] G. O'Reagan, *Introduction to the History of Computing: A Computing History Primer*. Springer, 2016.

[5] M. Gabbrielli ja M. Simone, *Programming Languages: Principles and Paradigms*. Springer London, 2010.

[6] M. Metcalf, "The Seven Ages of Fortran", English, *Journal of Computer Science and Technology*, vol. 11, nro 1, s. 1–8, huhtikuu 2011/04//. url: https://www.proquest.com/scholarly-journals/seven-ages-fortran/docview/2544434020/se-2.

[7] J. Backus, R. Beeber, B. S. et al., *Programmer's Reference Manual: The Fortran Automatic Coding System For the IBM 704 EDPM*. International Busi-

ness Machines Corporation, 1956. url: https://archive.computerhistory.org/resources/text/Fortran/102649787.05.01.acc.pdf (viitattu 28. 06. 2024).

[8] L. J. Kedward, B. Aradi, O. Čertík et al., "The State of Fortran", *Computing in Science & Engineering*, vol. 24, nro 2, s. 63–72, 2022. DOI: 10.1109/MCSE.2022.3159862.

[9] D. E. Knuth ja P. L. Trabb, "The Early Development of Programming Languages", tekninen raportti, 1976. url: https://archive.computerhistory.org/resources/text/Fortran/102679231.05.01.acc.pdf (viitattu 22. 07. 2024).

[10] J. E. Sammet, "The early history of COBOL", *SIGPLAN Not.*, vol. 13, nro 8, s. 121–161, elokuu 1978, ISSN: 0362-1340. DOI: 10.1145/960118.808378. url: https://doi.org/10.1145/960118.808378.

[11] Anonymous, "COBOL: Initial Specifications for a Common Business Oriented Language", tekninen raportti, 1960. url: http://www.bitsavers.org/pdf/codasyl/COBOL_Report_Apr60.pdf (viitattu 05. 07. 2024).

[12] M. Priestley, *A Science of Operations: Machines, Logic and the Invention of Programming*. Springer Verlag London Limited, 2011.

[13] A. N. Habermann, "Introduction to ALGOL 60 for those who have used other programming languages", syyskuu 1971. DOI: 10.1184/R1/6606608.v1. url: https://kilthub.cmu.edu/articles/journal_contribution/Introduction_to_ALGOL_60_for_those_who_have_used_other_programming_languages/6606608.

[14] D. E. Knuth, "A proposal for input-output conventions in ALGOL 60", *Commun. ACM*, vol. 7, nro 5, s. 273–283, toukokuu 1964, ISSN: 0001-0782. DOI: 10.1145/364099.364222.

[15] O.-J. Dahl ja K. Nygaard, ”SIMULA: an ALGOL-based simulation language”, *Commun. ACM*, vol. 9, nro 9, s. 671–678, syyskuu 1966, ISSN: 0001-0782. DOI: 10.1145/365813.365819.

[16] O.-J. Dahl, B. Myhrhaug ja K. Nygaard, ”Some features of the SIMULA 67 language”, teoksessa *Proceedings of the Second Conference on Applications of Simulations*, New York, New York, USA: Winter Simulation Conference, 1968, s. 29–31.

[17] K. Nygaard ja O.-J. Dahl, ”The development of the SIMULA languages”, *SIGPLAN Not.*, vol. 13, nro 8, s. 245–272, elokuu 1978, ISSN: 0362-1340. DOI: 10.1145/960118.808391.

[18] N. Wirth, ”Recollections about the development of Pascal”, teoksessa *The Second ACM SIGPLAN Conference on History of Programming Languages*, sarja HOPL-II, Cambridge, Massachusetts, USA: Association for Computing Machinery, 1993, s. 333–342, ISBN: 0897915704. DOI: 10.1145/154766.155378. url: https://doi.org/10.1145/154766.155378.

[19] N. Wirth, ”The Programming Language Pascal”, *Acta Inf.*, vol. 1, nro 1, s. 35–63, maaliskuu 1971, ISSN: 0001-5903. DOI: 10.1007/BF00264291.

[20] J. Buxton, ”CPL Elementary Programming Manual”, University of London, University of Cambridge, tekninen raportti, 1965. url: https://www.chilton-computing.org.uk/acl/pdfs/cpl.pdf (viitattu 25. 07. 2024).

[21] M. Richards, ”BCPL: a tool for compiler writing and system programming”, teoksessa *Proceedings of the May 14-16, 1969, Spring Joint Computer Conference*, sarja AFIPS ’69 (Spring), Boston, Massachusetts: Association for Computing Machinery, 1969, s. 557–566, ISBN: 9781450379021. DOI: 10.1145/1476793.1476880.

[22] D. M. Ritchie, ”The development of the C language”, teoksessa *The Second ACM SIGPLAN Conference on History of Programming Languages*, sarja HOPL-II, Cambridge, Massachusetts, USA: Association for Computing Machinery, 1993, s. 201–208, ISBN: 0897915704. DOI: 10.1145/154766.155580.

[23] T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney ja Y. Wang, ”Cyclone: A Safe Dialect of C”, teoksessa *2002 USENIX Annual Technical Conference (USENIX ATC 02)*, Monterey, CA: USENIX Association, kesäkuu 2002. url: https://www.usenix.org/conference/2002-usenix-annual-technical-conference/cyclone-safe-dialect-c.

[24] A. C. Kay, ”The early history of Smalltalk”, *SIGPLAN Not.*, vol. 28, nro 3, s. 69–95, maaliskuu 1993, ISSN: 0362-1340. DOI: 10.1145/155360.155364.

[25] D. Ungar ja R. B. Smith, ”Self: The power of simplicity”, *SIGPLAN Not.*, vol. 22, nro 12, s. 227–242, joulukuu 1987, ISSN: 0362-1340. DOI: 10.1145/38807.38828.

[26] D. of Defense, ”Department of Defense Requirements for High Order Computer Programming Languages. Tinman”, United States Department of Defense, tekninen raportti, 1976.

[27] J. Ichbiah, ”ADA: past, present, future”, *Commun. ACM*, vol. 27, nro 10, s. 990–997, lokakuu 1984, ISSN: 0001-0782. DOI: 10.1145/358274.358278.

[28] K. A. Loparo, C. H. Bull, P. B. Hansen et al., *The Programming Language Ada Reference Manual. Proposed Standard Document United States Department of Defense* (Lecture notes in computer science), en, 1980. painos. Berlin, Germany: Springer, huhtikuu 1981.

[29] B. Stroustrup, ”A history of C++: 1979–1991”, teoksessa *The Second ACM SIGPLAN Conference on History of Programming Languages*, sarja HOPL-

II, Cambridge, Massachusetts, USA: Association for Computing Machinery, 1993, s. 271–297, ISBN: 0897915704. DOI: 10.1145/154766.155375.

[30] S. Pemberton, "An Alternative Simple Language and Environment for PCs", *IEEE Software*, vol. 4, nro 1, s. 56–64, 1987. DOI: 10.1109/MS.1987.229797.

[31] F. Biancuzzi, *Masterminds of Programming: Conversations with the Creators of Major Programming Languages* (Theory in Practice (O'Reilly)). O'Reilly Media, 2009, ISBN: 9780596555504. url: https://books.google.fi/books?id=yB1WwURwBUQC.

[32] G. van Rossum. "SETL (was: Lukewarm about range literals)". (2000), url: https://mail.python.org/pipermail/python-dev/2000-August/008881.html (viitattu 12. 08. 2024).

[33] K. Kennedy ja J. Schwartz, "An introduction to the set theoretical language SETL", *Computers & Mathematics with Applications*, vol. 1, nro 1, s. 97–119, 1975, ISSN: 0898-1221. DOI: https://doi.org/10.1016/0898-1221(75)90011-5.

[34] B. Venners. "The Making of Python: A Conversation with Guido van Rossum, Part I". (2003), url: https://www.artima.com/articles/the-making-of-python (viitattu 13. 08. 2024).

[35] B. Venners. "Python's Design Goals: A Conversation with Guido van Rossum, Part II". (2003), url: https://www.artima.com/articles/pythons-design-goals (viitattu 13. 08. 2024).

[36] J. Gosling, B. Joy ja G. Steele, *The Java Language Specification*, L. Friendly, toim. Addison-Wesley, 1996.

[37] C. Sabharwal, "Java, Java, Java", *IEEE Potentials*, vol. 17, nro 3, s. 33–37, 1998. DOI: 10.1109/45.714612.

[38] B. W. Benson, ”JavaScript”, *SIGPLAN Not.*, vol. 34, nro 4, s. 25–27, huhtikuu 1999, ISSN: 0362-1340. DOI: 10.1145/312009.312023.

[39] B. Eich. ”Popularity”. (2008), url: https://brendaneich.com/2008/04/popularity/ (viitattu 14. 08. 2024).

[40] T. L. Aardsma, ”Java, JavaScript and JScript”, English, *Inside the Internet*, vol. 9, nro 3, s. 7, maaliskuu 2002, Copyright - Copyright Element K Journals Mar 2002; Last updated - 2014-05-22; CODEN - ININFR. url: https://www.proquest.com/magazines/java-javascript-jscript/docview/191109322/se-2.

[41] D. Merchant, ”Cross-platform JavaScript coding: Shifting sand dunes and shimmering mirages: SR”, English, *Library Computing*, vol. 18, nro 2, s. 141–142, 1999. url: https://www.proquest.com/scholarly-journals/cross-platform-javascript-coding-shifting-sand/docview/233611558/se-2.

[42] ECMA, ”ECMAScript: A general purpose, cross-platform programming language”, ECMA, tekninen raportti, 1997. url: https://ecma-international.org/wp-content/uploads/ECMA-262_1st_edition_june_1997.pdf (viitattu 14. 08. 2024).

[43] B. Abt, J. Ahto, A. Alexander et al. ”PHP Manual: Preface”. (ei pvm.), url: https://www.php.net/manual/en/preface.php (viitattu 14. 08. 2024).

[44] B. Abt, J. Ahto, A. Alexander et al. ”History of PHP”. (ei pvm.), url: https://www.php.net/manual/en/history.php.php (viitattu 14. 08. 2024).

[45] D. R. Naugler, ”C# 2.0 for C++ and Java programmer: conference workshop”, *J. Comput. Sci. Coll.*, vol. 22, nro 5, s. 1, toukokuu 2007, ISSN: 1937-4771.

[46] Anonymous, ”C# Language Specification”, ECMA, tekninen raportti, 2002. url: https://www.ecma-international.org/wp-content/uploads/ECMA-334_2nd_edition_december_2002.pdf (viitattu 20. 08. 2024).

[47] B. Cornelius. ”Java 5 catches up with C#”. (2005), url: https://www.barrycornelius.com/papers/java5/onefile/ (viitattu 14. 08. 2024).