hyväksymispäivä arvosana

arvostelija

Historiakatsaus assemblykääntäjistä korkean tason kielten kääntäjiin

Arttu Kilpinen

Helsinki 23.11.2016

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Arttu Kilpinen

Työn nimi — Arbetets titel — Title

Historiakatsaus assemblykääntäjistä korkean tason kielten kääntäjiin

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	23.11.2016	9 sivua

Tiivistelmä — Referat — Abstract

Ensimmäiset ohjelmointikielten kääntäjät, assemblykääntäjät, käänsivät symbolisille konekielille kirjoitettuja ohjelmia konekielisiksi ohjelmiksi. Täsmällisten kuvausjärjestelmien kehitys sekä koodin generoinnin teoria mahdollistivat tehokkaampien ohjelmointikielten kehityksen. Nykyisin käytössä olevat korkean tason ohjelmointikielet kehittyivät hiljalleen kuvausjärjestelmien kehittyessä ja syrjäyttivät symbolisella konekielillä ohjelmoinnin lähes kokonaan. Tässä dokumentissa käydään läpi historiallisia vaiheita symbolisten konekielten kääntäjistä nykyaikaisiin korkean tason ohjelmointikielten kääntäjiin. Läpi käydään useita merkittäviä ohjelmointikieliä ja niiden ominaisuuksia.

ACM Computing Classification System (CCS):
Software and its engineering -> software notations and tools -> Compilers

Avainsanat — Nyckelord — Keywords
Historia, Kääntäjät, Symbolinen konekieli, Ohjelmointikielet

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Sisältö

1 Johdanto

Kääntäjät ovat tietokoneohjelmia, jotka kääntävät lähdekielisen ohjelmakoodin kohdekieliseksi ohjelmaksi [Bauer, 1974]. Kohdekielenä on usein jonkin prosessoriarkkitehtuurin ymmärtämä konekieli.

Ohjelmointikielet sekä niitä ymmärtävät kääntäjät ja tulkit ovat keskeisessä asemassa ohjelmistotuotannossa. Kääntäjät mahdollistavat ohjelmien kirjoittamisen korkean tason ohjelmointikielillä sekä symbolisilla konekielillä, jotka puolestaan tekevät ohjelmoinnista nopeampaa sekä helpompaa. Niiden käyttäminen tekee ohjelmakoodista myös ymmärrettävämpää ja helpompilukuista. Yleisesti ottaen ohjelmointikielen ymmärrettävyys kasvaa abstraktiotason kasvaessa. Esimerkiksi matemaattisesti tutulla tavalla kirjoitetut aritmeettiset lausekkeet ovat ymmärrettävämpiä kuin vastaava symbolisella konekielellä ohjelmoituna. Lisäksi useat korkean tason ohjelmointikielet mahdollistavat — mikäli tarvittavat kääntäjät ovat olemassa — saman ohjelmakoodin käyttämisen useissa eri laitteistoissa sekä useilla eri käyttöjärjestelmillä. Koska eri laitteistoissa on erilaiset käskykannat, poistuu korkean tason ohjelmointikieliä käyttämällä myös tarve uudelleenohjelmoinnille.

Symboliset konekielet sekä korkean tason ohjelmointikielet ovat lähes yhtä vanhoja kuin ohjelmointikin. Ensimmäiset korkean tason ohjelmointikielten kääntäjät puolestaan ovat ohjelmointikieliä huomattavasti nuorempia, sillä ohjelmointikielten teoriaa kehitettiin vuosia ennen kuin ensimmäiset kääntäjät valmistuivat. Esimerkiksi ensimmäisenä korkean tason ohjelmointikielenä pidetty Plankalkül kehitettiin jo vuonna 1946, mutta sitä ymmärtävä kääntäjä valmistui vasta vuonna 1972 [Knuth and Pardo, 1976].

Ennen korkeatasoisille lausekielille kehitettyjä kääntäjiä oli pitkään käytössä vain symbolisia konekieliä ymmärtäviä ohjelmia, assemblykääntäjiä. Korkean tason ohjelmointikielten kehityttyä saatiin myös niitä tukevia kääntäjiä valmistettua. Vuonna 1952 valmistunut AUTOCODEn kääntäjä oli yksi ensimmäisiä kaupallisessa ohjelmistotuotannossa käytettyjä korkean tason kielen kääntäjiä [Knuth and Pardo, 1976].

2 Symbolinen konekieli ja assemblykääntäjät

Koska tietojenkäsittelytiede ei ole yhtä täsmällistä kuin matematiikka, ovat määritelmät usein vähemmän täsmällisiä ja saattavat poiketa toisistaan. Erään määritelmän mukaan assemblykääntäjä on kääntäjä, joka kääntää yksi yhteen symbolisella konekielellä kirjoitettuja komentoja konekielisiksi komennoiksi [Salomon, 1993]. Koska jokaisella laitteistolla on oma konekielensä ja tämä konekieli on myös ohjelmointikieli, pätee yleinen kääntäjien määritelmä myös assemblykääntäjiin. Lähdekielenä assemblykääntäjän ymmärtämä symbolinen konekieli tarkoittaa konekieltä, jossa laitteen ymmärtämät binääriset konekäskyt on korvattu ihmisille helpommin muistettavilla sanoilla eli symboleilla.

2.1 Historia ensimmäisistä assemblykääntäjistä

Koska ennen ensimmäisiä assemblykääntäjiä ei ollut mitään ohjelmointia helpottavia työkaluja, tuli ne ohjelmoida suoraan konekoodilla kuten yksi ensimmäisiä, vuonna 1949 valmistunut EDSAC tietokoneen assemblykääntäjä toteutettiin. Vaikka korkean tason kielten kääntäjät alkoivat kehittyä lähes heti ensimmäisten assemblykääntäjien valmistuttua, pysyi symbolisilla konekielillä ohjelmointi pitkään suosiossa. Alkuun korkean tason kielten automaattista käännöstyötä pidettiin lähinnä teoreettisena tutkimisena ja käytännössä kaikki ohjelmoijat uskoivat ettei automaattisesta koodin generoinnista tule ikinä tarpeeksi tehokasta oikeaan ohjelmointiin [Knuth and Pardo, 1976].

Vaikka nykyaikaiset korkean tason ohjelmointikielien kääntäjät tuottavat hyvin optimoitua koodia, on hyvän ohjelmoijan kirjoittama symbolinen konekieli silti lähes poikkeuksetta parempaa. Tämän takia symbolisia konekieliä käytetään vielä nykyisin matalan tason ohjelmoinnin lisäksi suurta laskentatehoa vaativien ohjelmien optimointiin. Ennen symbolisten konekielten kehitystä ohjelmointi tapahtui kirjoittamalla laitteistoriippuvaista tietyn prosessorin ymmärtämää binäärikoodia. Siitä huolimatta, että käskykannat olivat nykyiseen verrattuna suhteellisen yksinkertaisia, oli ohjelmointi hidasta ja työlästä. Tietokoneiden kehittyessä ja ohjelmien monimutkaistuessa tarve ohjelmointikielille kasvoi. Symboliset konekielet kehitettiin varhain ja nykyisin lähes kaikki sovellusohjelmat kirjoitetaan korkean tason ohjelmointikielillä.

2.2 Assemblykääntäjien toiminnasta ja toteutuksesta

Assemblykääntäjät ymmärtävät jotakin symbolista konekieltä ja osaavat tuottaa tästä linkkerin avulla konekielisen suoritettavan ohjelman. Symboliset konekielet ovat matalan tason laiteriippuvaisia ohjelmointikieliä joille on tyypillistä lähes täysi koodin symbolisuus. Tämä tarkoittaa sitä, että suurin osa ohjelmakoodista on käännettävissä yksi yhteen laitteiston ymmärtämän konekielen kanssa. Poikkeuksen tekee kuitenkin ohjelman osoitteina käytettävät tunnisteet (label), joiden arvot assemblykääntäjä voi vapaasti päättää. Tunnisteina voi toimia joko paikat ohjelman koodiosassa tai muuttujina käytetyt muistipaikat. Symbolisen konekielen avainsanat ovat siis symboleja laitteiston ymmärtämälle binäärille. Konekielellä on mahdollista kirjoittaa suoraan suorittimen rekistereihin. Tämä tekee symbolisilla konekielillä ohjelmoimisesta yhtä laiteläheistä kuin suoraan konekielillä ohjelmointikin. Laiteläheisyys puolestaan tekee ohjelmista laitteistoriippuvaisia, sillä eri suorittimilla voi olla erilaiset käskykannat. Symbolien käyttäminen vähentää huomattavasti kirjoitusvirheiden määrää ja tekee koodista helpomman kirjoittaa ja käydä läpi. Tunnisteiden käyttö puolestaan poistaa tarpeen muistaa muuttujien sekä konekäskyjen osoitteita.

Kuva 1 selventää symbolisten konekielten ennalta määrättyjen symbolien sekä ohjelmoijan määrittelemien tunnisteiden eron. Esimerkkikoodi on TTK91 [Ttk, 1991] virtuaaliprosessorille tehty ohjelma, joka tulostaa käyttäjälle luvut 0...5. Keltaisella pohjalla olevat symbolit ovat niin sanottuja tunnisteita, joilla voi olla eri arvo käännöskerrasta ja kääntäjästä riippuen. Kaikki harmaalla pohjalla oleva koodi käännetään siis yksi yhteen.

```
        LOAD    R1, = 0
        STORE   R1,   X
LOOP    LOAD    R1, = 5
        COMP    R1,   X
        JGRE          END
        LOAD    R1,   X
        OUT     R1, = CRT
        ADD     R1, = 1
        STORE   R1,   X
        JUMP          LOOP
END     NOP
```

Kuva 1: TTK91 esimerkkikoodi

3 Historiaa korkean tason kielten kääntäjistä

Korkena tason ohjelmointikielellä tarkoiteteen ohjelmointikieltä, jossa kielen sekä siitä käännettävän konekielen välillä on selkeä abstraktiotaso ja kääntäminen edellyttää muutakin, kuin mekaanista sanojen vaihtamista ennalta määrättyjen sääntöjen perusteella. Tämän määritelmän perusteella korkean tason ohjelmointikielillä tarkoitetaan niitä kieliä, jotka eivät ole symbolisia konekieliä.

3.1 Täsmällisten kuvausjärjestelmien kehitys

Tietojenkäsittelytieteilijät ovat jo tietokoneiden alkuajoista lähtien yrittäneet kuvailla ohjelmien suoritusta ja algoritmeja konekieltä abstraktimmalla tasolla. Alan Turingin julkaisussa vuonna 1936 esitettiin määritelmä tietojenkäsittelijöiden hyvin tuntemasta laskentalaitteesta, turingin koneesta. Laitteen yhteydessä määriteltiin myös matemaattinen esitystapa, jolla sen toimintaa voitiin täsmällisesti kuvailla. Vaikka esitystapa oli vaikea eikä kyseisiä Turingin esittelemiä laitteita ollut kuin teoreettisella tasolla, Turingin esitystapa edusti kehittyneintä formaalia kuvausta, 'kieltä', mitä siihen aikaan oli olemassa.

Toisen maailmansodan jälkeen vuonna 1945 saksalainen Konrad Zuse aloitti oman tietokoneohjelmien kuvailuun tarkoitetun kielen Plankalkülin kehittämisen. Zusen sanoin Plankalkülin tarkoitus oli luoda puhtaasti formaali esitystapa mille tahansa laskentaongelmalle. Tässä hän onnistuikin varsin hyvin. Plankalkülissa voidaan määritellä aritmetiikan ja ohjausrakenteiden lisäksi rajaton määrä sisäkkäisiä tietorakenteita ja Zusen työhön viitataankin usein ensimmäisenä korkean tason ohjelmointikielenä. Vaikka kyseessä oli huomattavan edistyksellinen järjestelmä, se ei kuitenkaan vaikuttanut ohjelmointikielten kehitykseen juuri lainkaan. Zusen artikkelit julkaistiin vasta vuonna 1972 muiden, kehittyneempien kielten jo olemassa ollessa. Vaikka Plankalkülille toteutettiinkin kääntäjä, ei sitä juuri koskaan käytetty koska silloin oli jo Plankalkülia huomattavasti kehittyneempiä ohjelmointikieliä.

Samoihin aikoihin Zusen kanssa Yhdysvaltalaiset Herman Goldstine ja John von Neumann koittivat ratkaista samaa ongelmaa. Heidän ratkaisunsa algoritmien ja tietokoneohjelmien kuvaamiseen oli varsin erilainen. Von Neumann ja Goldstine esittivät ratkaisuksi virtausdiagrammia (flow diagram), esitystapaa jossa ohjelmat kuvataan nuolien ja laatikoiden avulla.

Vuonna 1946 Marylandissa työskennellyt Haskell B. Curry kehitti ENIAC tietoko-

neelle aikaansa nähden monimutkaista ohjelmaa. Curryn työ ENIACIN parissa sai hänet ehdottamaan formalismia ohjelmistojen toiminnalle. Hänen formalisminsa perustui uuteen ajatukseen ohjelman suorituksen lohkomaisesta rakenteesta, mitä hän nimitti divisiooniksi. Divisioonien tulisi olla rakennettu niin että niiden laskenta olisi toisistaan riippumatonta. Tämän voisikin rinnastaa esimerkiksi C-kielen paikallisiin tietorakenteisiin ja käännösyksiköihin perustuvaan suoritukseen. Curryn formalismi oli kuitenkin hieman luonnoton sillä suoritusyksiköillä oli useita lopetuskohtia sekä nykykielistä poiketen useita aloituskohtia. Historiallisesti työ oli kuitenkin merkittävä, sillä se sisälsi algoritmeja joilla kuvauksesta pystyttiin tuottamaan konekoodia. Näitä rekursiivisia — vaikkakin toteuttamatta jääneitä — algoritmeja voidaankin pitää ensimmäisinä koodin generointiin tarkoitettuina algoritmeina.

3.2 Kohti ensimmäisiä kääntäjiä

Millekään aiemmin mainituista ohjelmointikielistä ei tähän mennessä oltu toteutettu kääntäjiä. Ne toimivat ohjelmoijien käsitteellisenä apuna auttaen ohjelmien suunnittelussa, mutta jättäen toteutuksen ihmisille. Tästä huolimatta olivat ne kaikki merkittäviä askeleita kohti parempia ohjelmointikieliä sekä niiden kääntäjiä. Ilman täsmällisiä esitystapoja ei koodin generointi ikinä olisi tullut mahdolliseksi.

Ensimmäinen korkean tason ohjelmointikieli, jolle toteutettiin tulkki oli Short Code. Sitä kehitti John W. Mauchly vuonna 1949 ja William F. Schmitt toteutti sille tulkin. Tulkki toimi alkuun BINAC tietokoneella mutta se ohjelmoitiin myöhemmin myös UNIVACille. Yksityiskohtia Short Coden toiminnasta ei ikinä julkaistu, joten sen tarkemmasta toiminnasta ei ole tietoa. Vuonna 1955 julkaistusta ohjelmoijille tarkoitetussa manuaalissa kerrotaan kuitenkin kuinka ohjelmaa voidaan käyttää. Short Code oli siis algebrallinen tulkki, joka osasi suorittaa aritmeettisia laskutoimituksia ilman konekielistä ohjelmointia. Ohjelma luki syötettä ja suoritti vastaavat toiminnot ajetulla laitteistolla.

1950-luvun alussa Heiniz Rutishauser ja Corrado Böhm työskentelivät Zürichin teknillisessä yliopistossa Sveitsissä. Vaikka he työskentelivät samassa paikassa ja saman aiheen parissa, eivät he työskennelleet yhdessä. Rutishauser julkaisi 1952 artikkelin, jossa hän kuvasi hypoteettisen tietokoneen sekä siinä toimivan kääntäjän kehittämälleen ohjelmointikielelle. Julkaisu oli merkittävä, sillä siinä kuvattiin ensimmäistä kertaa menetelmä kääntäjien toteuttamisesta sekä koodin generoinnista.

Rutishauserin kollega Corrado Böhm kehitti myös ohjelmointikieltä sekä tämän

kääntäjää. Hänen julkaisunsa oli Rutishauserin julkaisua vieläkin merkittävämpi, sillä hän oli toteuttanut kääntäjän tämän omalla kielellä. Böhmin kieli ei kuitenkaan osannut käsitellä muita kuin positiivisia kokonaislukuja, joten sen käyttöarvo jäi melko pieneksi. Kääntäjien teorian kehityksen kannalta se oli kuitenkin korvaamaton. Böhmin kääntäjä kykeni tarkistamaan koodin syntaksia lineaarisessa ajassa kun Rutishauserin kääntäjä toimi suuruusluokassa n^2. Lisäksi Böhmin kääntäjä hallitsi matemaattisten operaattoreiden sidontajärjestyksen, sekä osasi käsitellä sulkeita aritmeettisissa lausekkeissa. Lisäksi Böhm oli ensimmäinen tietojenkäsittelijä, joka todisti matemaattisesti ohjelmointikielensä voivan laskea minkä tahansa laskettavan funktion.

Vaikka Rutishauser ja Böhm olivat kumpikin valmistaneet omat kääntäjänsä, pidetään ensimmäisenä 'oikeana' kääntäjänä silti Alick E. Glennien 1952 valmistamaa AUTOCODE ohjelmistoa. Aiemmista kääntäjistä poiketen AUTOCODE toteutettiin oikealle laitteistolla ja sen tuottama konekieli oli oikeasti suoritettavissa. AUTOCODEa pystyttiin siis käyttämään oikeiden, käyttökelpoisten ohjelmien tekemiseen.

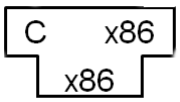
Vuoden 1954 alussa John Backus rupesi kehittämään kokoamansa kehittäjätiimin kanssa automaattisen ohjelmoinnin järjestelmää. Järjestelmän oli tarkoitus olla hyvin kehittynyt, joten suureksi haasteeksi muodostui järjestelmän saaminen tarpeeksi tehokkaaksi. Loppuvuodesta 1954 kehittäjäryhmä julkaisi suunnitelman järjestelmästä 'The IBM Mathematical FORmula TRANstating system' — FORTRAN. Kuten jo aiemmin oli todettu, tehokkaan koodin tuottaminen ei ollut lainkaan helppoa. Ryhmän julkaisu alkoikin painottamalla sitä tosiasiaa, että FORTRAN oli tehokas. Aiemmin ohjelmoijien tuli valita helpon ohjelmoinnin ja hitaan suorituksen tai työlään ohjelmoinnin ja nopean suorituksen väliltä, mutta FORTRANin tarjoaisi parhaat puolet molemmista [IBM, 1954]. FORTRAN 0 dokumentti esittää myös ensimmäisen yrityksen esittää ohjelmointikielen syntaksi täsmällisesti. Tätä voidaan pitää Backuksen myöhemmin esittelemän kielioppimuodon Backus Naur Formin (BNF) edeltäjänä.

Kun FORTRAN kaksi ja puoli vuotta myöhemmin saatiin toteutettua, oli se aikansa tehokkain sekä monipuolisin ohjelmointikieli. FORTRAN tuotti kohtuullisen tehokasta koodia ja keittäjät sanoivat sen olevan lähes yhtä tehokasta kuin hyvän ohjelmoijan kirjoittama symbolinen konekieli. FORTRANissa oli myös paljon ominaisuuksia, joita ei oltu aiemmin nähty. Se oli esimerkiksi ensimmäinen ohjelmointikieli, jossa muuttujien nimet voivat olla useamman merkin pituisia.

Ensimmäisen julkaisun jälkeen FORTRANissa oli kuitenkin useita ognelmia. Bugeja oli paljon ja eräs FORTRANIN kehittäjistä, Saul Rosen, sanoikin ettei uskonut FORTRANin ikinä tulevan toimimaan [Rosen, 1964]. Vaikeuksista huolimatta FORTRANista tuli hyvin suosittu ja sitä käytetiin enemmän kuin oltiin osattu odottaa.

4 T-kaaviot

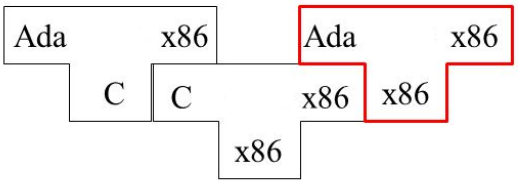
Kääntäjien suunnittelussa ja mallintamisessa on käytetty useita erilaisia kaavioita. Käytetyin ja tunnetuin lienee Harvey Bratmanin 1961 ehottama kääntäjää kuvaava kaavio [Bratman, 1961], josta käy ilmi kääntäjän ymmärtämä kohde- ja lähdekieli sekä kieli, jolla kääntäjä toimii. Kaaviota kutsutaan Bratman-kaavioksi tai T-kaavioksi. Jälkimmäinen nimi tulee kaavion muodosta, jossa T-kirjaimen muotoisessa alueessa vasen pääty ketoo lähdekielen, oikea pääty kohdekielen ja alasakara kertoo millä kielellä kääntäjä toimii.



Kuva 2: T-kaavio, joka kuvaa x86 arkkitehtuurilla toimivaa c-kääntäjää, jonka kohdekieli on x86 konekieli

Kaavioita toisiinsa liittämällä voidaan havainnollistaa monimutkaisiakin toimintaketjuja, joita kääntäjät suorittavat. Oletetaan että käytössä on c-kielellä kirjoitettu kääntäjä, joka kääntää ada-kieltä x86 konekielelle. Lisäksi käytössä on edellisen kuvan esimerkissä oleva x86 arkkitehtuurilla toimiva c-kääntäjä, jonka kohdekieli on x86 konekieli. Näiden kahden kääntäjän avulla voidaan tuottaa x86 alustalla toimiva ada kääntäjä, jonka kohdekieli on x86. Kahden ensimmäisen kääntäjän yhteistyöllä saadaan siis kolmas kääntäjä. Huomattavaa on, että prosessin alimman tasoinen kääntäjä toimii aina jossakin todellisessa laitteistossa, eikä täten voi olla muu kuin jonkin laitteiston ymmärtämä konekieli.

T-kaavio oli melko yksinkertainen, joten siitä on kehitetty paranneltuja vaihtoehtoja. Jay Earley ja Howard Sturgis laajensivat Bratmanin kaaviota lisäämällä siihen omi-



Kuva 3: c:llä kirjoitetun ada-kääntäjän ja x86:lla toimivan c kääntäjän avulla voidaan tuottaa x86:lla toimiva ada-kääntäjä

naisuuksia. Alkuperäisen T-kaavion kuvatessa vain kääntäjiä, Earleyn ja Sturgisin kaavioissa pysyi kuvaamaan myös tulkkeja. Lisäksi kaavioon sisällytettiin suoritusta kuvaava osa sekä sovellusohjelmaa kuvaava kaavio. Heidän kaavioissaan määritellään seuraavan kuvan mukaiset elementit.

5 Yhteenveto

Tietojenkäsittelytieteessä kääntäjä tarkoittaa ohjelmaa joka kääntää lähdekielisen ohjelmakoodin kohdekieliseksi ohjelmakoodiksi. Koska ennen muun kuin konekielten kehitystä ohjelmointi tapahtui suoraan laitearkkitehtuurin ymmärtämällä muodolla, ei tarvetta kääntäjille ollut. Koska konekoodin ohjelmointi oli varsin työlästä, kehitettiin avuksi symbolisia konekieliä, joissa tietyt binäärijonot oli korvattu paremmin muistettavilla tekstuaalisilla symboleilla.

Korkeamman tason ohjelmointikielet kehittyivät ohjelmoijien tarpeesta kuvata ohjelmistojen toimintaa korkeammilla abstraktiotasoilla. Täsmälliset kuvausjärjestelmät kehitettiin alunperin ilman ajatusta kääntäjistä taikka automaattisesta koodin generoinnista. Vaikka kuvausjärjestelmien sekä koodia generoivien algoritmien kehitys oli alkuun vain teoreettista tutkimista, huomattiin potentiaali niiden tehokkaaseen käyttöön varsin pian. Tämän jälkeen ohjelmointikieliä ruvettiin kehittämään varta vasten automaattisen koodin generoinnin takia ja ensimmäiset oikeasti hyödylliset kielet sekä niiden kääntäjät kehitettiin.

Taulukko 1 tiivistää kappaleessa 2 esitettyjen ohjelmointikielten merkittävimmät

Taulukko 1: Yhteenveto ohjelmointikielten ja kuvausjärjestelmien kehityksestä

Kieli	Kehittäjä	Ensimmäinen
Plankalkül	Zuse	Ohjelmointikieli, Hierarkinen data
Virtauskaaviot	Goldstine, Von Neumann	Hyväksytty ohjelmointimetodologia
Short Code	Mauchly	Toteutettu korkean tason ohjelmointikieli
Formules	Böhm	Samalla kielellä kirjoitettu kääntäjä
AUTOCODE	Glennie	Käyttökelpoinen kääntäjä
FORTRAN I	Backus	I/O formaatti, kommentit, globaali optimointi

piirteet. Lisäksi taulukossa esitetään ohjelmointikielten nimet sekä päätekijät.

Lähteet

[Ttk, 1991] (1991). Ttk91 reference. `https://www.cs.helsinki.fi/group/titokone/ttk91_ref_en.html`. Accessed: 2016-10-21.

[Bauer, 1974] Bauer, F. L. (1974). *Compiler Construction – An Advanced Course*. SpringerÂVerslag, Berlin Heidelberg GmbH.

[Bratman, 1961] Bratman, H. (1961). An alternative form of the 'uncol' diagram.

[Goldstine and Von Neumann, 1947] Goldstine, H. and Von Neumann, J. (1947). Planning and coding problems for an electronic computing instrument.

[IBM, 1954] IBM (1954). Specifications fot the ibm mathematical formula translating system, fortran Programming Research Group, I.B.M Applied Science Div.

[Knuth and Pardo, 1976] Knuth, D. and Pardo, L. (1976). *The early development of programming languages*. Stanford University.

[Rosen, 1964] Rosen, S. (1964). Programming systems and languages, a historical survey.

[Salomon, 1993] Salomon, D. (1993). *Assemblers and Loaders*. Ellis Horwood Ltd.