

Anna Whitlocks Gymnasium

Teknikprogrammet inriktning design och produktutveckling

Läsåret 2021/2022

Handledare: Sofie Danielsson

Ett internationellt programmeringsspråk

Elias Sjögren

Abstract

Computer programming has gone from being a relatively unusual skill to something more and more people are learning, for jobs and as a hobby. The global demand for programmers is increasing in line with the internet and global digitalization. Most programming languages are however written in and use English while the majority of the world does not use English as its primary language. This report examines the possibility of creating a specification and prototype implementation of a programming language which is translateable between multiple different written languages as a way of making programming more accessible to a global audience.

Innehållsförteckning

Inledning	3
Bakgrund	4
Begreppslista	4
Programmeringsspråk	5
En global efterfrågan för programmerare	5
Block- och flödesprogrammering	5
Textprogrammering	6
Symbolprogrammering	7
Ett programmeringsspråks uppbyggnad	7
Lexikalanalys	8
Syntaxanalys	8
Formell grammatik	8
Kompilation, transpilation och interpretation	9
Syfte och Frågeställning	11
Metod	12
Val av programmeringsspråkstyp	12
Verktyg och implementationsspråk	12
Planering och struktur	12
Genomförande	13
Specifikation	13
Värden	13
Identifierare	13
Kommentarer	13
Makron	14
Operatorer	14
Nyckelord	14
Uttryck	15
Satser	16
Program	17
Implementation	17
Transpilation	17
Resultat	18
Definition	18
Exempel	18
Exempelprogram “Hej, Världen!”	19
Diskussion och Slutsats	20
Mellanspråklig användning	20
Icke-latinska skriftsätt	20
Läsbarhet	20

Kontext och nyckelord	21
Globala målen och språkets användbarhet	21
Vidare utveckling	21
Källförtäckning	22
Bilagor	24
Bilaga 1. Diagram av en formell grammtik för heltals aritmetik	24
Bilaga 2. Full lexikal EBNF specifikation av programmeringsspråket	25
Bilaga 3. Full syntax EBNF specifikation av programmeringsspråket	26
Bilaga 4. Exempelprogram uttryck	27
Bilaga 5. Exempelprogram satser	28

Inledning

Datorprogrammering har gått från att vara en relativt ovanlig färdighet till något fler och fler människor lär sig, för jobb och som en hobby. Industrin och den globala efterfrågan för programmerare ökar i takt med internet och i princip all teknologi, från din mikrovågsugn och dator till det komplexa system som styr exempelvis elnätet. Detta är inte heller något lokalt fenomen, digitaliseringen är global men majoriteten av programmeringsspråken som är ett väsentligt verktyg för denna teknologiska utveckling är fortfarande begränsade av en språkbarriär då de flesta programmeringsspråk använder sig av engelskan. Denna rapport undersöker möjligheten att skapa en specifikation samt prototyp av ett programmeringsspråk med mål att vara översättnings- och användbart mellan olika skriftspråk.

Bakgrund

Begreppslista

Program eller datorprogram är vilket som helst program som en dator kan tolka och genomföra. Detta innefattar maskinkod men även källkod och olika typer av plattformsoberoende kod.

Programmeringsspråk eller programspråk är ett skriftligt, formellt språk för att skriva källkod. Ofta är ett programmeringsspråk i sig ett datorprogram som kan översätta kod från något som är läsbart av en människa till något en dator kan köra.

Maskinkod är den binära kod, det vill säga de ettor och nollor, som representerar olika instruktioner som en dator direkt kan tolka och genomföra.

Virtuell maskin är en virtuell dator som i programmeringssyfte används för att skapa en miljö där plattformsoberoende kod eller olika mellanrepresentationer kan köras så som om dess kod var äkta maskinkod.

Plattformsoberoende kod eller mellanrepresentation är en typ av kod som kan köras på en virtuell maskin på datorn istället för att köras direkt på datorn genom maskinkod.

Källkod eller källprogram är ett datorprogram representerat som det programmeringsspråk det från början var skrivet i.

Dator eller datorprocessorn kan, likt en kock som följer ett recept eller en chaufför som följer en karta tolka och köra olika instruktioner som den har fått, dessa instruktioner är maskinkod.

Interpretator eller programtolk är ett datorprogram som tolkar och genomför de instruktioner som beskrivs i programmet direkt utan översättning eller kompilation till maskinkod eller någon mellanrepresentation.

Kompilator är ett datorprogram som kan översätta ett program skrivet i källkod till datorprogram som går att köra representerade som maskinkod eller olika mellanrepresentationer.

Integrerad utvecklingsmiljö är ett datorprogram som innehåller ett antal olika verktyg för att arbeta med exempelvis programmering. Detta kan inkludera exempelvis en textredigerare, visuell programmeringsmiljö och en kompilator eller interpretator.

Datastruktur är en abstraktion av flera olika datavärden så som exempelvis en lista, ett träd eller en tabell. Detta görs för att datorn effektivt skall kunna arbeta med icke-primitiva datavärden och för att förenkla användandet.

Programmeringsspråk

I grunden kan alla datorprocessorer tolka vissa instruktioner vid namn maskinkod ("maskinkod", 2021). Allt en dator gör bygger på dessa, ofta mycket enkla instruktioner, representerade binärt i datorns minne. Maskinkoden kan vara svåra att läsa och skriva som människa. Komplexiteten för program skrivna i maskinkod snabbt ökar för större program, detta på grund av maskinkodens närhet till hårdvaran som tolkar programmet. För att tackla detta problem så har olika programmeringsspråk skapats för att göra programmeringen av datorer mer abstrakt och därmed möjliggjort mer avancerad, effektivare och enklare programmering (Aho, 1986).

Källkod är den centrala delen i ett programmeringsspråk som representerar det som ett datorprogram gör och är skriven för hand. Vanligtvis består källkoden av olika kodord, operationer, uttryck och värden i en text. Dessa kodord, operationer, uttryck och värden tolkas sedan på olika sätt av programmeringsspråket för att till vara körbart av en dator som maskinkod (Wikipedia contributors, 2021b).

När man talar om programmeringsspråk talar man ofta om olika hög- eller lågnivåspråk. Detta är ett koncept som används för att definiera hur nära källkoden är till den slutgiltiga maskinkoden. Historiskt tidiga programmeringsspråk är ofta närmare maskinkod och skulle därmed kunna kallas låga medan moderna är mer abstraherade, det vill säga mer abstrakta än den faktiska maskinkoden som körs och därmed är högnivåspråk. Fördelen med lågnivåspråk är just dess närhet till hur en dator fungerar och tänker vilket gör att det går att skriva mycket effektiv kod, ett bra exempel på ett sådant språk är assembler ("assemblerspråk", 2021) vilket är ett samlingsnamn för språk där maskininstruktioner är översatta till ord och värden för olika hårdvara. Till skillnad från lågnivåspråk är högnivåspråk mycket enklare för utvecklaren att skriva samt förstå och är de vanligast använda (Carbonnelle, 2021; *TIOBE Index*, 2021).

En global efterfrågan för programmerare

Programmering som yrke har under de senaste hundra åren växt fram otroligt snabbt och behovet för mjukvaruutvecklare är idag mycket stort. I endast Sverige beräknas det saknas omkring 70000 personer för att möta det växande behovet inom IT branshen (*IT-Kompetensbristen*, 2020) till år 2024. År 2021 uppskattades det finna omkring 26,9 miljoner mjukvaruutvecklare globalt, ett antal som uppskattas öka till kring 45 miljoner år 2030 (*How Many Developers Are There In The World In 2021?*, 2021).

I en årlig undersökning av internet- och mjukvaruutvecklingsforumet Stack Overflow har man frågat 80000 utvecklare, studenter och andra användare om bland annat utbildning, land, erfarenhet och andra relevanta frågor (*Stack Overflow Developer Survey 2021*, 2021). I undersökningens statistik om vart undersökningsdeltagarna bor är det endast kring 27,31%¹ som bor i länder där engelska är ett officiellt förstaspråk. Av dem mer än 37 olika programmeringsspråken som det deltagarna i undersökningen har använt sig av som svar i olika frågor är endast ett språk² ej baserat på engelskan.

Att programmeringsspråk inte vanligtvis är skrivna i användarens modersmål utan på engelska skulle kunna påverka spridningen, utbildningen och användningen av programmeringsspråk i icke engelskspråkiga länder. I detta fall skulle det framförallt vara det fjärde globala målet som definieras som säkerställandet av en inkluderande och likvärdig utbildning av god kvalitet och främja livslångt lärande för alla ("Mål 4", 2022; "Sustainable development goals", 2022a). Det åttande globala målet anständiga arbetsvillkor och ekonomisk tillväxt skulle kunna vara relevant ("Mål 8", 2022; "Sustainable development goals", 2022b) där fokus framförallt ligger på målets verkan för inkluderande och ekonomisk tillväxt, något som gått att koppla till den globala efterfrågan för programmerare.

Block- och flödesprogrammering

I ett programmeringsspråk där översättning till andra språk prioriteras finns det ett antal olika tillvägagångssätt, alla med olika för- och nackdelar.

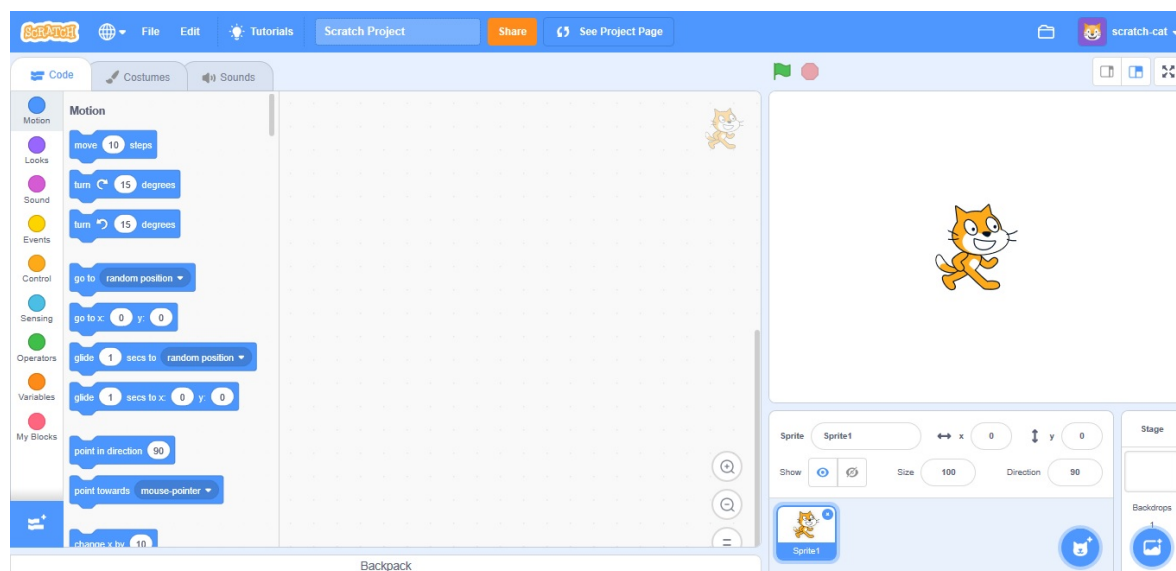
¹Inräknat i denna beräkning är USA med 18,33%, Storbritannien med 5,37% samt Kanada med 3,61%.

²Programmeringsspråket APL

Blockprogrammering är bland det vanligaste av de olika typerna av översättningsbara programmeringsspråk. Dessa språk är ofta till för utbildningssyfte och fungerar på sätt att de består av block, liknande Lego. Blocken kan ha olika funktioner som till exempel värden, uttryck eller olika satser. Man bygger ett program genom att koppla ihop dessa block i en specialbyggd programmeringsmiljö. De kändaste och mest använda programmeringsspråket (*Scratch - Statistics*, 2021) implementerat på detta vis är Scratch (*Scratch - About*, 2021), originellt utvecklat av MIT senare Scratch Foundation. Detta språk utvecklades i utbildningssyfte med översättning och lättanvändning i åtanke och dess målgrupp är framförallt grundskolebarn.

Flödesprogrammeringsspråk är på många sätt lika de block baserade programmeringsspråk i det att dem använder sig av visuella block. I ett flödesbaserat språk är dock dessa block inte sekventiella utan ihopkopplade med sladdar som representerar ett värdes flöde genom programmet. Spelmotorer som Unreal (*Blueprint Visual Scripting*, 2021), Unity (*Bolt Visual Scripting* | *Unity*, 2021) och Godot (*Godot Engine contributors*, 2021) samt 3d modelleringsprogram som Blender (*Introduction — Blender Manual*, 2021) använder sig av nodbaserad programmering för ett enklare alternativ till traditionell textbaserad programmering. Detta alternativ är likt blockprogrammering i det att etiketterna och texterna på noderna är översättningsbara.

Något som alla dessa block- och flödesprogrammeringsspråk har gemensamt är dess användning av visuella programmeringsmiljöer där man på olika sätt kombinerar byggstenarna för respektive språk, något som inte traditionellt finns för textbaserade programmeringsspråk.



Figur 1: Programmeringsspråkets Scratches visuella programmeringsmiljö

Textprogrammering

Ett annat alternativ till visuella programmeringsspråk så som de block- och flödesbaserade språken tidigare nämnda är ett mer traditionellt textbaserat språk. Det finns många textbaserade programmeringsspråk är att dem flesta är av den typen men väldigt få uppfyller kravet att vara översättningsbart. Vad som menas med ett textbaserat programmeringsspråk är ett språk varav programmen består av vanliga tecken och bokstäver men som följer en viss grammatik och syntax, likt hur naturliga språk också följer en grammatik.

Ett tidigt exempel på ett översättningsbart programmeringsspråk är det av ALGOL 68 (*Wikipedia contributors*, 2021a). Detta språk standardiserades (*Van Wijngaarden m.fl.*, 1968) och dess standard publicerades i flera olika naturliga språk. Denna standard översattes till ryska, tyska, franska, bulgariska, kinesiska och senare japanska samt en för dess användning i blindskrift. Detta ledde till att standarden antogs och accepterades av både UNESCOs organisation IFIP samt Sovjets och senare Rysslands standardorganisation.

Citrine är ett programmeringsspråk där lokalisering är en av kärnfunktionerna vilket har lett till dess översättning till 111 olika naturliga språk. Språket lokaliserar nyckelord, nummer och skiljetecken. För att konvertera mellan olika språk kan användaren själv översätta programmet men inget inbyggt verktyg i programmet verkar göra detta åt användaren. Detta då alla olika naturliga språk som Citrine stödjer publiceras som separata program utan vetskapen om hur man skulle översätta ett program från ett till ett annat naturligt språk (Mooij (Gabor), 2021).

Skriftspråk	Exempelkod
Engelska	<code>write: 'Hello World', stop.</code>
Svenska	<code>skriva: 'Hej Världen', sluta.</code>
Tyska	<code>schreiben: 'Hallo Welt', stop.</code>

Scheme (*The Scheme Programming Language*, 2003) är ännu ett standardiserat programmeringsspråk med möjlighet till internationalisation. Detta är dock inte en kärnfunktion i språket utan har istället utvecklats av användare som har använt språkets flexibilitet för att skapa ett bibliotek ("metaphorm", 2021) där olika översättningar finns. Eftersom olika språk kan laddas in dynamisk går det att skriva flerspråkiga program i Scheme. Detta leder dock till den nackdelen att programmeringsspråket ej enkelt kan översättas då flera olika språk skulle kunna existera i samma program samt det faktum att programmeringsspråket inte är byggt med översättning, lokalisering eller internationalisation som en kärnfunktion.

Skriftspråk	Exempelkod
Engelska	<code>(display "Hello World")</code>
Svenska	<code>(visa "Hej Världen")</code>
Tyska	<code>(anzeige "Hallo Welt")</code>

Symbolprogrammering

Det sista typen som är relevant som ett alternativ för att skapa ett översättningsbart eller internationellt programmeringsspråk är den av vad jag väljer att kalla symbolprogrammeringsspråk. Detta då den använder sig av symboler istället för nyckelord vilket leder till att det inte är bundet till ett naturligt språk. Ett exempel på ett sådant språk är exempelvis APL (Iverson, 1962) men även helt vanlig matematisk notation (Helmenstine, 2019).

En kritik av dessa språk är dess dåliga anpassning till användning digitalt vilket i matematisk notation fall har lett till ett antal olika alternativ för att skriva ut matematik på datorn så som det traditionellt görs på papper (*ISO/IEC 40314:2016*, 2016; Wikibooks, 2022; Wikipedia contributors, 2022). Denna kritik kan även dras till APL där avancerade symboler som inte lätt går att skriva på ett vanligt tangentbord används vilket leder till svårigheter att både läsa och skriva för någon som inte redan har bekantat sig med språkets symboler (Wiki, 2022).

Ett programmeringsspråks uppbyggnad

Ett programmeringsspråk är egentligen i sig ett datorprogram vars uppgift är att översätta en viss inmatning till ett program som datorn kan köra. Detta görs genom att dela upp uppgiften i ett antal olika delar. Det kan se olika ut för olika språk och tillvägagångssätt men generellt delar man in det i tre större delar: lexikalanalys ("lexikalanalys", 2021), syntaxanalys ("syntaxanalys", 2021) och till sist kompilation ("kompilator", 2021) eller interpretation ("interpretator", 2021). Man har programmerat dessa delar i ett annat³ programmeringsspråk vilket har skapat ett program som kan genomföra till exempel lexikalanalys, syntax analys och till sist kompilation för att sedan producera det slutgiltiga programmet.

³Eller samma, se Wikipedia contributors (2021c)

Lexikalanalys

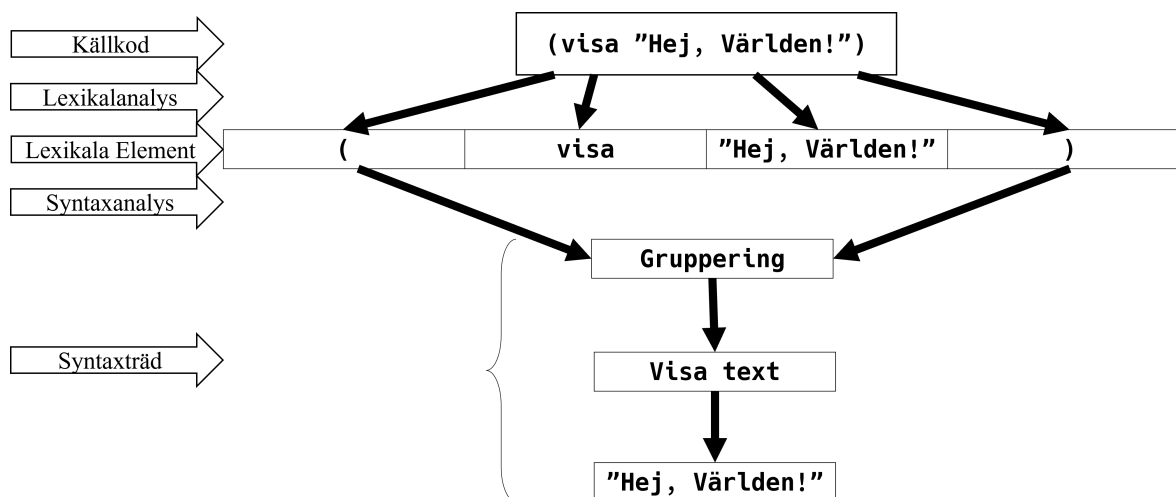
Lexikalanalysen är oftast första steget i programmet och delar upp och klassificerar källkoden till en lista av lexikala element så som nyckelord, värden, operatorer och skiljetecken. Exempelvis skulle en sträng så som "(visa "Hej Världen")" delas upp i följande element:

Elementtyp	Värde
Skiljetecken	(
Nyckelord	visa
Sträng	Hej Världen
Skiljetecken)

Syntaxanalys

För att konvertera de lexikala elementen till något som datorn effektivt kan använda och "förstå" krävs ytterligare analys, så kallad syntaxanalys. Detta innebär vanligtvis att dessa lexikala element analyseras utifrån en formell grammatik för att producera en datastruktur som representerar programmets uppbyggnad samt validerar språkets grammatik.

Vanligtvis så skapar syntaxanalysen ett syntaxträd som representerar programmets uppbyggnad på ett mer konkret sätt än en lista av lexikala element. Detta "träd" byggs upp utav olika syntaxnoder där varje nod är som en förgrening alternativt slutet på en gren i ett träd.



Figur 2: Fullständiga processen för att skapa ett slutgiltigt syntaxträd för exempel källkoden "(visa "Hej Världen")"

Formell grammatik

Ett formellt språk eller system defineras som en delmängd av en ändlig uppsättning av tecken (i många fall är denna uppsättning av tecken de lexikala element). Vad som ingår i denna delmängd är definierat av det formella språkets grammatik, en så kallad formell grammatik. Grammatiken kan vara beskrivas på flera olika sätt, bland annat genom ett formellt system så som EBNF, kort för Extended Backus-Naur-form Pattis (2015) eller informellt genom en skriftlig beskrivning eller instruktion.

Grammatiken av exempelvis ett formellt språk som beskriver matematisk heltals aritmetik skulle kunna definieras av följande: en form av EBNF definition samt en informell skriftlig definition som förklarar EBNF definitionen. Även en visuell representation finns i bilaga 1.

```
siffra    = "0" | "1" | "2" | "3" | "4"
           | "5" | "6" | "7" | "8" | "9"
```

```

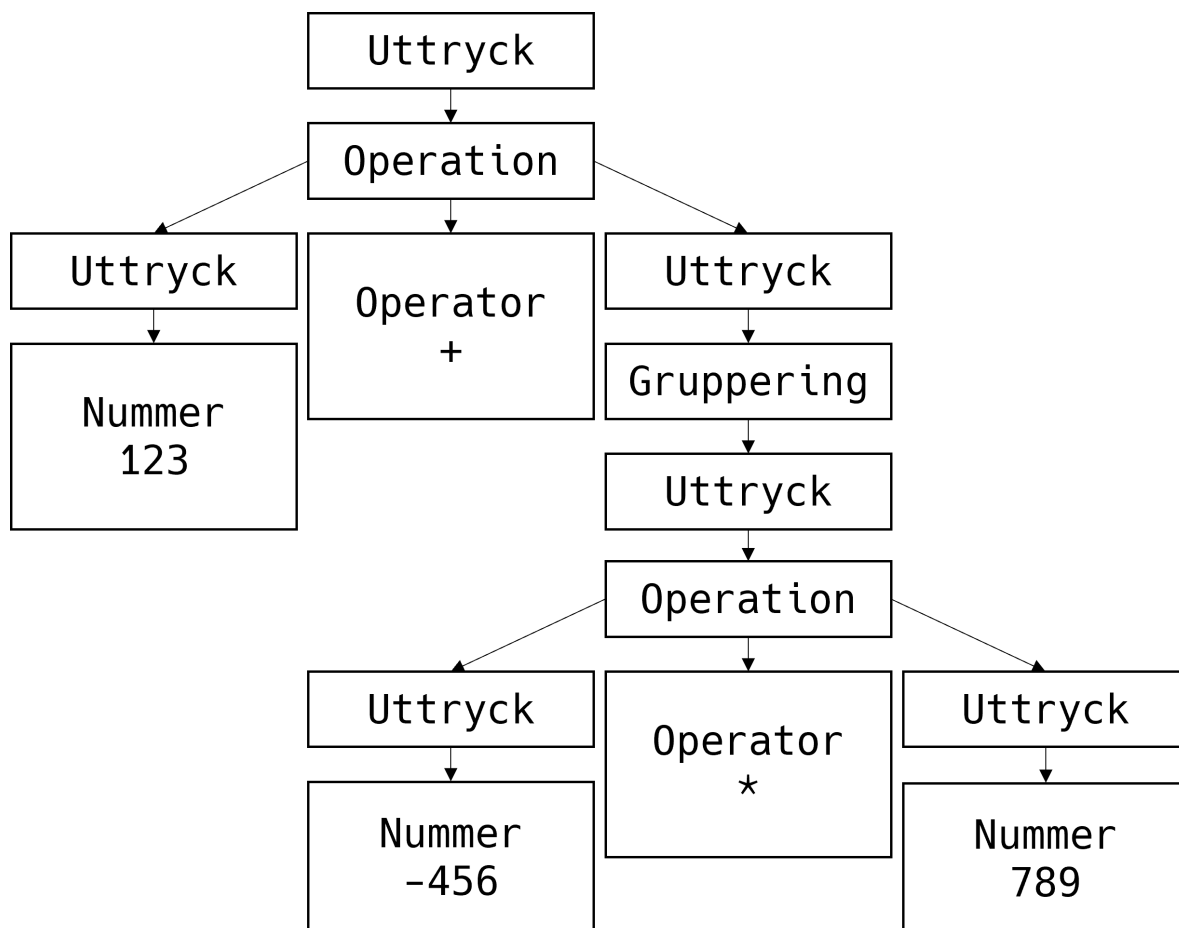
nummer      = "-"? siffra+
operator    = "+" | "-" | "x" | "÷"
operation   = uttryck operator uttryck
gruppering  = "(" uttryck ")"
uttryck     = nummer | operation | gruppering

```

Ovan definition går att beskriva som följande med ord:

- En **siffra** defineras som någon utav tecknen för siffrorna 0 till 9
- Ett **nummer** defineras som först ett valfritt minus tecken följt utav en eller flera siffror
- En **operator** defineras som ett plus, minus, gånger eller divisionstecken
- En **operation** defineras som ett uttryck följt av en operator och sedan ett till uttryck
- En **gruppering** defineras som ett uttryck inom paranteser
- Ett **uttryck** defineras som antingen ett nummer, operation eller gruppering

Med hjälp av denna definition går det att analysera en lista av tecken eller lexikala element för att bygga ett syntaxträd. Exempelvis skulle ett uttryck som “123 + (-456 * 789)” skapa det syntaxträd som finns i figur 3. Hur man genomför denna analys finns det ett antal olika sätt men vanligtvis delar man in syntaxanalysmetoderna i två familjer: “top-down” respektive “bottom-up” metoder (Lunell, 1991).



Figur 3: Syntaxträd för källkoden “123 + (-456 * 789)” analyserad med hjälp av ovan definierad formell grammatik

Kompilation, transpilation och interpretation

Syntaxträdets uppgift är att av en kompilator, transpilator eller interpretator tolkas för att konvertera källkoden av programmet till maskinkod eller direkt köras av datorn ifall det interpreteras.

Sättet syntaxträdet tolkas på för alla dessa fall är genom att “vandra” över trädets syntaxnoder. Exempelvis skulle ett träd som det i 3 börja med det som först skulle konverteras till maskinkod eller interpreteras, det vill säga först grunduttrycken: “123”, “-456” och “789”. Dessa nummer skulle då läggas till i programmets maskinkod, transpilationsmål eller tolkas i interpretatorn, sedan skulle operationen “*” utföras på dem två senaste värdena i programmet för att sedan utföra operationen “+”. Detta skulle då resultera i ett program som räknar ut aritmetiken eller ett värde beroende på om källkoden kompileras eller interpreteras.

Syfte och Frågeställning

Målet med projektet är att utveckla ett prototypprogram samt specifikation av ett programmeringsspråk. Denna prototyp skall vara översättningsbart och lättförståeligt på olika språk. Syftet med detta är för att undersöka möjligheten och olika tillvägagångssätt för att skapa ett programmeringsspråk som går att översätta till olika språk och fortfarande vara enkelt att förstå.

Metod

Val av programmeringsspråkstyp

I valet av programmeringsspråkstyp finns det flera olika alternativ som på olika sätt uppnår det konstaterade målet.

Problemet som uppkommer för både flödes- och blockbaserad programmering är det att dom kräver en integrerad utvecklingsmiljö så som en hemsida för Scratch eller respektive programs skrivbordsapplikation för Unreal, Unity, Godot och Blender. Att utveckla en integrerad utvecklingsmiljö utöver ett programmeringsspråk och dess beståndsdelar skulle utgöra ytterligare problem och är egentligen utanför projektets omfattning.

Verktyg och implementationsspråk

Det programmeringsspråk som valdes som implementationsspråk för prototyp implementationen av arbetet var TypeScript (Bierman m.fl., 2014), ett JavaScript baserat språk med tillägget av explicita datatyper. Anledningen till detta val var en avvägning mellan simplicitet, abstraktion, prestanda och möjligheten till plattformsoberoende kod för att göra språket körbart även i en webbläsare. Stöd för körning utav programmeringsspråket i både webbläsare och som program görs genom ett verktyg och körtidsmiljö vid namn Deno (Dahl, 2018).

Planering och struktur

Programmeringsspråkets lexikal-, syntax- och semantiskaanalys planerades och specificerades med hjälp av EBNF. Även en kompilator till JavaScript samt en översättare planerades. JavaScript valdes som kompilationsmål på grund av dess liknande struktur till det planerade språket samt dess stöd på dem flesta datormiljöerna.

Översättaren planerades fungera endast på lexikal nivå, det vill säga den är kontextfri och ej bryr sig om exempelvis ordning eller elementtyp. Detta går eftersom den endast planeras översätta lexikalelement av typen “nyckelord” som vanligtvis är kontextfria⁴.

⁴Se underrubriken Kontext och nyckelord för vidare reflektion

Genomförande

Specifikation

En specifikation som beskriver programmeringsspråkets syntax och grammatik skapades i EBNF format, delar av denna specifikation är dock beroende utav det skriftspråk som önskas användas för språkets nyckelord. Denna specifikation finns beskriven samt definerad i följande underrubriker.

Värden

De värdetyper som valdes till specifikationen var "boolesk", "nummer", "sträng" och "inget". Detta är endast ett litet urval utav de traditionella värdetyperna i ett programmeringsspråk för att avgränsa specifikationens och implementationens komplexitet. Dessa fyra värdetyper är grunden till alla algoritmer, uttryck och satser som programmeringsspråket kan processera. EBNF representationen är följande:

```
boolesk      = sant_nyckelord | falsk_nyckelord

siffra       = "0" | "1" | "2" | "3" | "4"
              | "5" | "6" | "7" | "8" | "9"
nummer       = siffra+ "." {siffra+}

sträng_värde = valfri_karaktär*
sträng       = "'" (sträng_värde - "\"") "'"

värde        = boolesk | nummer | sträng | inget_nyckelord
```

Identifierare

En identifierare är en textuell referens eller ett namn till en funktion, variabel eller konstant. Detta skulle exempelvis kunna vara variabel- och funktionsnamn inom matematiken så som "x" eller "pi". Detta språk definerar en identifierare som en sekvens som börjar med en karaktär i unicode kategorin bokstav följt utav något av unicode kategorierna bokstav, symbol eller nummer.

```
identifierare = unicode_bokstav
              (unicode_bokstav
               | unicode_symbol
               | unicode_nummer)*
```

Kommentarer

Kommentarer är inom programmeringsspråket är ett sätt för användaren att kommentera kod med text som ignoreras utav implementationen. Dessa är definerade på samma sätt som i många andra språk så som C, JavaScript och Rust. Det finns två typer av kommentarer: en-rads- och fler-rads-kommentarer. En-rads-kommentarer börjar där användaren har skrivit "//" och slutar vid radens slut medan fler-rads-kommentarer börjar vid "/*" och slutar vid "*/". Båda dessa kommentarstyperna ignorerar allt inom dessa avgränsare.


```

kommentar          = en_rads_kommentar | fler_rads_kommentar
en_rads_kommentar  = "//" valfri_karaktär* ny_rad
fler_rads_kommentar = "/*" valfri_karaktär* "*/"

```

Makron

Ett makro används i programmeringsspråket som ett sätt att kommunicera med kompilatorn för att exempelvis importera kod skriven i andra programmeringsspråk än detta. Likt kommentarer finns två olika typer av makron, en-rads-makron och fler-rads-makron. Dessa består utav en typidentifierare och ett värde. Typidentifieraren är till för att tipsa kompilatorn om hur makrots värde skall tolkas. Exempelvis genom att importera makrots värde som obearbetad kod.

```

makro              = en_rads_makro | fler_rads_makro
makro_identifierare = valfri_karaktär*
makro_värde        = valfri_karaktär*
en_rads_makro      = "#(" makro_identifierare ")" makro_värde ny_rad
en_rads_makro      = "#(" makro_identifierare ")" "{" makro_värde "}"

```

Operatorer

Operatorerna som är definierade för språket agerar endast på nummer- och booleskvärden på samma sätt som dess motsvarigheter i matematiken.

Symbol	Beskrivning
=	Anger en variabls värde
+	Addition vid operationer med två nummervärde, annars för att markera att ett nummervärde är positivt
-	Subtraktion vid operationer med två nummervärde, annars för att markera att ett nummervärde är negativt
*	Multiplikation av två nummervärden
/	Division av två nummervärden
%	Restprodukten av två nummervärden vid division
==	Jämför två värden för likhet
!=	Jämför två värden för olikhet
<	Jämför ifall det första nummervärdet är mindre än det andra nummervärdet
<=	Jämför ifall det första nummervärdet är mindre eller lika med än det andra nummervärdet
>	Jämför ifall det första nummervärdet är större än det andra nummervärdet
>=	Jämför ifall det första nummervärdet är större eller lika med än det andra nummervärdet
!	Inverterar en boolesk, dvs gör om sanna booleskvärden till falska och falska till sanna
	Boolesk eller operation som tar två booleskvärden och returnerar sant ifall något av värdena är sanna
&&	Boolesk eller operation som tar två booleskvärden och returnerar sant ifall båda värdena är sanna

EBNF definitionen är densamma som symbolerna i tabellen.

Nyckelord

Nyckelorden i programmeringsspråket används för att markera olika uttryck och satser. Dessa nyckelord är dock skriftspråksspecifika vilket gör att språkets definition ändras beroende på skriftspråk.

Engelska	Svenska	Tyska	Franska
function	funktion	funktion	function
return	returnera	rückkehr	retourne
if	om	ob	si

Engelska	Svenska	Tyska	Franska
while	medan	während	pendant
break	avbryt	abbrechen	casse
continue	fortsätt	fortsetzen	continue
variable	variabel	variable	variable
constant	konstant	konstante	constant
none	inget	null	rien
true	sant	wahr	vrai
false	falskt	falsch	faux

EBNF definitionen är densamma som orden i kolumnen för det skriftspråk man använder.

Uttryck

Binära- och unärauttryck

Binära och unärauttryck är en grupp av olika operationer som utförs på ett, två eller flera uttryck. Denna typ utav uttryck kräver en operator och defineras som följande:

```
binärt_uttryck = uttryck binär_operator uttryck
unärt_uttryck  = unär_operator uttryck
```

Dessa uttryck är exempelvis addition, eller negation utav andra uttryck: "a + b" eller "-123" där uttrycket är kombinationen av ett eller två underliggande uttryck samt en operator.

Villkorsuttryck

Villkorsuttryck används för att i uttryck välja antingen ett uttryck eller det andra beroende på om ett villkor (booleskt uttryck) är sant eller falskt likt en om-sats.

```
villkor          = uttryck
uttryck_om_sant  = uttryck
uttryck_om_falskt = uttryck
villkors_uttryck = om_nyckelord villkor
                  uttryck_om_sant
                  annars_nyckelord
                  uttryck_om_falskt
```

Funktionsanrop

Funktionsanrop är likt funktioner i matematiken anrop till tidigare definierade funktioner, dessa anrop kan ta noll eller flera uttryck som argument.

```
funktions_argument = uttryck
funktions_anrop    = identifierare
                    "(" funktions_argument
                    ("," funktions_argument)* ")"
```

Variabelåtkomst

Variabel- och konstantåtkomst görs genom att specificera variabeln eller konstantens identifierare.

```
variabel_åtkomst = identifierare
```

Gruppering

En gruppering används i ett uttryck för att markera vilken ordning ett uttryck utförs, likt hur i matematiken paranteser används för prioritering. Det är även värt att notera att även operatorerna följer prioriteringsreglerna.

```
gruppering = "(" uttryck ")"
```

Sammanfattning av uttryck

Sammanfattningsvis kan ett uttryck i programmeringsspråket definieras som något av ovan definierade kategorierna eller ett värde.

```
uttryck = unärt_uttryck
        | binärt_uttryck
        | vilkors_uttryck
        | funktions_anrop
        | variabel_åtkomst
        | gruppering
        | värde
```

Satser

Villkorssats

Vilkors-, om-, eller if-satsen är en konstruktion i programmeringsspråket för att programmera logik och kontrollera flödet av ett program. En if-sats består av ett uttryck som representerar villkoret för att köra dess inre satser. Valfritt är även en annars- eller else-sats som körs ifall villkorsuttrycket ej är sant.

```
villkor      = uttryck
vilkors_sats = om_nyckelord "(" villkor ")" sats
              (annars_nyckelord sats)?
```

Medan, fortsätt och avbryt

För mer avancerade program finns medan- eller while-satsen med dess tillhörande nyckelord: **“fortsätt”** och **“avbryt”**. Denna sats finns likt en villkorssats för att kontrollera flödet av ett program men till skillnad från en villkorssats fortsätter medansatsen att köra dess inre satser tills villkoret är falskt.

```
villkor      = uttryck
medan_sats   = medan_nyckelord "(" villkor ")" sats
```

Returnera

En funktion finns som en abstraktion för att återanvända och abstrahera exempelvis algoritmer. För att funktioner ska kunna användas i uttryck och skapa nya värden finns nyckelordet **“returnera”** för att returnera ett värde från en funktion och avbryta funktionens körning.

```
returnera = returnera_nyckelord uttryck
```

Variabel- och konstantdeklaration

För att deklarera variabla och konstanta värden som kan användas i en lokal räckvidd, det vill säga exempelvis det nuvarande kodblocket eller funktionen.

```
variable_deklaration = variabel_nyckelord
                     identifierare "=" uttryck
konstant_deklaration = konstant_nyckelord
                     identifierare "=" uttryck
```

Kodblock

Ett kodblock består av en lista utav satser som alla existerar i kodblockets egna räckvidd. Detta gör att variabler och konstanter, men även värden som är skapta i det lokala kodblocket inte kan användas av något på en högre nivå, som exempelvis ifall kodblocket skulle vara inuti ett annat kodblock.

```
kod_block = "{ sats* }"
```

Sammanfattning av satser

Satser är sammanfattningsvis alla ovanstående rubriker samt uttryck och makron. Dessa satser används inne i exempelvis funktioner för att skapa logiken som utgör ett program.

```
sats = villkors_sats
      | medan_sats
      | returnera
      | variable_deklaration
      | konstant_deklaration
      | kod_block
      | uttryck
      | makro
```

Program

Ett program är en lista utav syntaxelement på toppnivån. De enda elementen som räknas som detta är funktionsdeklarationer. Funktioner är till för att abstrahera och binda ihop sammanhängande satser av kod som lätt kan användas flera gånger i koden.

```
funktions_argument    = identifierare
funktions_deklaration = funktion_nyckelord
                        identifierare
                        "(" funktion_argument
                        "(" funktion_argument)* ")"
                        kod_block

program               = funktions_deklaration
```

Implementation

Implementationen av programmeringsspråket gjordes i programmeringsspråket TypeScript. Programmetts källkod delades upp i lexikalanalys, syntaxanalys och transpilation enligt de ovan presenterade rubrikerna och implementerades i den ordningen. Sedan skapades även ett program för att använda programmeringsspråket med hjälp av en terminal eller kommandtolk.

Specifikationen skrevs samtidigt som programmeringen utav både lexikalanalysen och syntaxanalysen för att underlätta processen och därmed anpassa specifikationen för att inte överkomplicera programmeringen.

Transpilation

Transpilationen av programmeringsspråket görs till målspråket JavaScript, ett språk som matchar ganska nära till det specificerade programmeringsspråkets syntax och funktion. Exempelvis konverteras funktioner skrivna i språket till den dess motsvarighet i JavaScript.

Resultat

Implementationen utav programmeringsspråket publiceras kontinuerligt som öppen-källkod på github.com/eliassjogreen/gyarb. Resulterande program kan köras lokalt för att utföra lexikalanalys, syntaxanalys, kompilation eller översättning med följande kommandon i en terminal eller kommandotolk:

```
$ gyarb tokenize  --language [en|sv|de|fr] <file>
$ gyarb parse     --language [en|sv|de|fr] <file>
$ gyarb compile   --language [en|sv|de|fr] <file>
$ gyarb run       --language [en|sv|de|fr] <file>
```

Det resulterande programmeringsspråket utvecklat utifrån de principer och idéer presenterade i bakgrunden och frågeställningen stödjer fyra skriftspråk med hjälp av en abstraktion av lexikalanalysen som tillåter att modulärt byta ut nyckelorden. Detta tillåter användningen av samtliga stödda skriftspråken samt en singular syntexanalysator och kompilator oberoende av den inmatade källkoden.

Definition

Programmeringsspråket definierades i definitionsformatet EBNF och dess fulla lexikala och syntaxspecifikation kan finnas i bilaga 3 respektive 4. Dessa två specifikationer är uppdelade för att reflektera programmets interna uppdelning.

Exempel

För att demonstrera hur programmeringsspråket ser ut, dess läsbarhet och översättningar till olika språk se bilagorna 5 och 6 och även i filkatalogen **src/examples** i källkoden. Dessa exempel behandlar värden, operationer och funktioner samt algoritmer och makron.

Exempelprogram “Hej, Världen!”

Nedan defineras källkoden för ett exempelprogram vars mål är att skriva texten “Hej, Världen!” på respektive skriftspråk i kommandotolken. För att exekvera programmet körs det kommando som defineras längst ned i kodblocket under de kommande underrubrikerna. Resultatet som sedan skrivs ut i konsolen defineras under kommandot samt källkoden längst upp i respektive språks exempel. Samma mönster kan även användas för att köra exemplen definerade i bilagorna 5 och 6. För att istället för att köra koden utföra lexikalanalys, syntaxanalys eller kompilation ersätts nyckelordet “run” i kommandot respektive nyckelord för den önskade handlingen. Det vill säga “tokenize”, “parse” eller “compile”.

Engelska

```
function entry() {  
    print("Hello, World!")  
}  
  
$ gyarb run --language en src/examples/hello_world/hellow_world.en.gy  
Hello, World!
```

Svenska

```
funktion ingång() {  
    skriv("Hej, Världen!")  
}  
  
$ gyarb run --language se src/examples/hello_world/hellow_world.se.gy  
Hej, Världen!
```

Tyska

```
funktion eingang() {  
    schreiben("Hallo, Welt!")  
}  
  
$ gyarb run --language de src/examples/hello_world/hellow_world.de.gy  
Hallo, Welt!
```

Franska

```
function entrée() {  
    écrivez("Bonjour le monde!")  
}  
  
$ gyarb run --language fr src/examples/hello_world/hellow_world.fr.gy  
Bonjour le monde!
```

Diskussion och Slutsats

Det framtagna programmeringsspråket behandlar vissa av de idéerna som presenterades i bakgrunden och uppfyller frågeställningen, men lämnar även många viktiga funktioner och anpassningar som skulle önskas för ett mer kapabelt programmeringsspråk.

Mellanspråklig användning

I det utvecklade programmeringsspråket går det i nuläget att använda delar av ett program eller ett programbibliotek skrivet i ett skriftspråk i ett annat program skrivet i ett annat skriftspråk. Bristen i detta är flera, bland annat att identifierare inte översätts på grund av det faktum att dem är arbiträra och inte konsekvent kan översättas ifall användaren inte specifikt förser varje identifierare med en översättning för varje skriftspråk programmet är implementerat för.

Detta skulle vara möjligt med hjälp av en ny grammatik eller genom användningen av makron. Ett annat alternativ är användningen av en översättningstjänst för att på så sätt automatiskt översätta identifierare. Detta kräver dock vidare undersökning då ett automatiserat system skulle skapa en mängd nya problem, bland annat felöversättningar, försämrad läsbarhet och identifierare som inte går att översätta. Det skulle även göra att språket inte är deterministiskt om inte översättningssystemet är deterministiskt.

Icke-latinska skriftsätt

En viktig funktion för vidare internationalisering utav ett programmeringsspråk som det utvecklat i denna rapport är även stöd för icke latinska skriftsätt, exempelvis arabiska eller kinesiska. Här presenterar sig även problem med symboler och dess lokala betydelse samt ifall språket inte skrivs från vänster till höger som så är fallet med just arabiska. Detta presenterar ett antal utvecklingspunkter då språksätt varierar mycket och inte lätt kan generaliseras till en och samma standard. Komplexiteten med att hantera olika skriftspråk kan även ses i olika digitala översättningssystem för webbsidor, applikationer och i självaste standarden för digital text: unicode.

Läsbarhet

Ännu en punkt den här rapporten inte undersöker är språkets läsbarhet och hur lättanvänt det är. Något som är mycket relevant utifrån det kontext att det inte nödvändigtvis är vilket språk nyckelorden är skrivna på som spelar roll för ett programmeringsspråks läsbarhet. Detta kan det finnas flera olika förklaringar bakom bland annat att på grund av att det är så få nyckelord i de flesta programmeringsspråk och att dem få som finns ofta är liknande eller likadana. Detta skulle gå att vidare undersöka då det uppenbarligen finns programmerare inte talar eller skriver engelska men ändå programmerar i språk med engelska nyckelord. Vidare går det även att undersöka om programmeringsspråk lärs in och används som naturliga språk. Även intressant är att undersöka ifall inläringen av ett programmeringsspråk anpassat för ens modersmål är annorlunda ifrån dem traditionella programmeringsspråken som utgår ifrån engelskan.

Kontext och nyckelord

Dem flesta språk böjer ord beroende på kontext, exempelvis kontext som genus, konjugation eller numerus. Detta kontext skulle då kunna ändra även nyckelordens korrekta form beroende på funktion- eller variabelnamn.

Globala målen och språkets användbarhet

Det fjärde globala målet, säkerställandet av en inkluderande och likvärdig utbildning av god kvalitet och främja livslångt lärande för alla är det framföralla mål som detta projekt utvecklades i åtanke med. Projektet utvecklades för att vara lättförståeligt och användbart i flera olika språk, dock utan någon som helst kvantitativ bakgrund till hur man gör ett programmeringsspråk eller användargränssnitt lättförståeligt eller effektivt översättningsbart.

Detta går att koppla till ovanstående rubrik läsbarhet då det är mycket möjligt att ett programmeringsspråk som skrivs i ens modersmål inte är enklare utan kontraintuitivt svårare på grund av det stora antal redan etablerade lärare, programmerare och material där programmeringsspråket som används är engelska baserat.

Vidare utveckling

För att vidare utveckla studieområdet skulle en kvantitativ studie utföras. Studien skulle exempelvis kunna mäta hur individer upplever språkets läsbarhet , korrekthet eller lärande enkelhet i olika skriftspråk. Man skulle även kunna studera en fler-språkig individs möjlighet att förstå programmeringsspråkets olika skriftspråksvarianter.

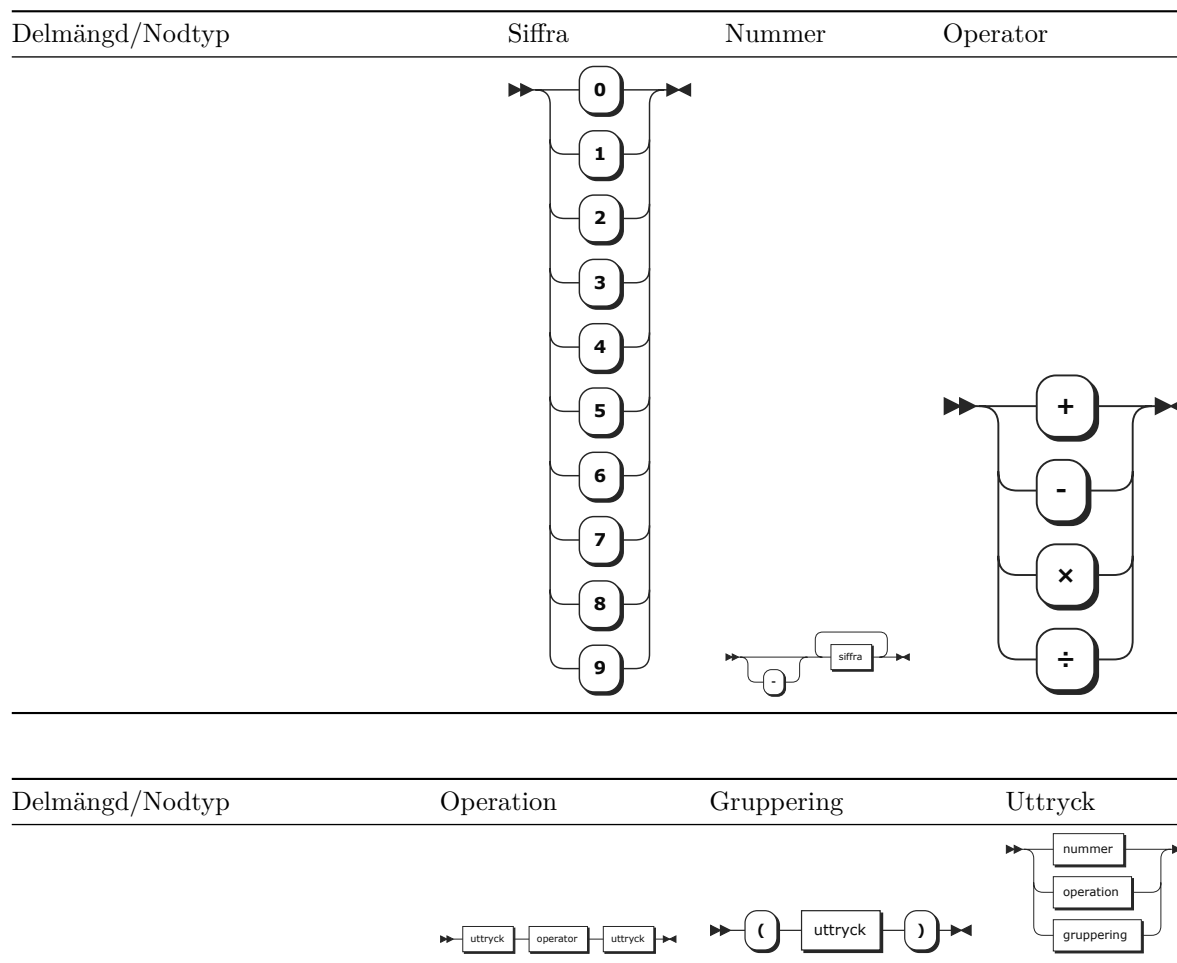
Källförtäckning

- Aho, A. (1986). *Compilers, principles, techniques, and tools*. Addison-Wesley Pub. Co.
- assemblerspråk. (2021). I *Nationalencyklopedin*. <https://www.ne.se/uppslagsverk/encyklopedi/lång/assemblerspråk>
- Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding typescript. *European Conference on Object-Oriented Programming*, 257–281.
- Blueprint Visual Scripting*. (2021). Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints>
- Bolt Visual Scripting | Unity*. (2021). Unity Technologies. <https://unity.com/products/unity-visual-scripting>
- Carbannelle, P. (2021). *PYPL PopularitY of Programming Language index*. <https://pypl.github.io/PYPL.html>
- Dahl, R. (2018). *10 Things I Regret About Node.js*. Youtube. <https://youtu.be/M3BM9TB-8yA>
- Godot Engine contributors. (2021). *VisualScript*. https://docs.godotengine.org/en/stable/getting_started/scripting/visual_script/index.html
- Helmenstine, A. M. (2019). Why Mathematics Is a Language. I *ThoughtCo*. <https://www.thoughtco.com/why-mathematics-is-a-language-4158142>
- How Many Developers Are There In The World In 2021?* (2021). Future Processing. <https://www.future-processing.com/blog/how-many-developers-are-there-in-the-world-in-2019>
- interpretator. (2021). I *Nationalencyklopedin*. <http://www.ne.se/uppslagsverk/encyklopedi/lång/interpretator>
- Introduction — Blender Manual*. (2021). Blender Foundation. https://docs.blender.org/manual/en/latest/render/shader_nodes/introduction.html
- IT-Kompetensbristen*. (2020). IT&Telekom företagen. <https://www.techsverige.se/2020/12/it-kompetensbristen/>
- Iverson, K. E. (1962). *A Programming Language*. John Wiley & Sons, Inc.
- kompilator. (2021). I *Nationalencyklopedin*. <http://www.ne.se/uppslagsverk/encyklopedi/lång/kompilator>
- lexikalanalys. (2021). I *Nationalencyklopedin*. <http://www.ne.se/uppslagsverk/encyklopedi/lång/lexikalanalys>
- Lunell, H. (1991). *Kompilatorkonstruktion i teori och praktik*. Studentlitteratur.
- maskinkod. (2021). I *Nationalencyklopedin*. <http://www.ne.se/uppslagsverk/encyklopedi/lång/maskinkod>
- ”metaphorm”. (2021). *international-scheme*. <https://github.com/metaphorm/international-scheme>
- Mooij (Gabor), G. J. G. T. de. (2021). *Localized Programming Language Citrine*. <https://citrine-lang.org>
- Mål 4: God utbildning för alla - Globala Målen. (2022). I *Globala Målen*. Globala Målen. <https://www.globalamalen.se/om-globala-malen/mal-4-god-utbildning-alla/>
- Mål 8: Anständiga arbetsvillkor och ekonomisk tillväxt - Globala Målen. (2022). I *Globala Målen*. Globala Målen. <https://www.globalamalen.se/om-globala-malen/mal-8-anstandiga-arbetsvillkor-och-ekonomisk-tillvaxt/>
- Pattis, R. E. (2015). *EBNF: A Notation to Describe Syntax*.
- Plastics – Determination of fracture toughness – Linear elastic fracture mechanics (LEFM) approach* (Vol. 2000). (2016). [Standard] (Vol. 2000). International Organization for Standardization. <https://www.iso.org/standard/58439.html>
- Scratch - About*. (2021). Scratch Foundation. <https://scratch.mit.edu/about>

- Scratch - Statistics*. (2021). Scratch Foundation. <https://scratch.mit.edu/statistics>
- Stack Overflow Developer Survey 2021*. (2021). Stack Overflow. <https://insights.stackoverflow.com/survey/2021>
- Sustainable development goals: United Nations Development Programme. (2022a). I *UNDP*. UNDP. <https://www.undp.org/sustainable-development-goals#quality-education>
- Sustainable development goals: United Nations Development Programme. (2022b). I *UNDP*. UNDP. <https://www.undp.org/sustainable-development-goals#decent-work-and-economic-growth>
- syntaxanalys. (2021). I *Nationalencyklopedin*. <http://www.ne.se/uppslagsverk/encyklopedi/lång/syntaxanalys>
- The Scheme Programming Language*. (2003). <https://groups.csail.mit.edu/mac/projects/scheme>
- TIOBE Index*. (2021). TIOBE Software. <https://www.tiobe.com/tiobe-index/>
- Van Wijngaarden, A., Mailloux, B. J., Peck, J., & Koster, C. H. A. (1968). Draft Report on the Algorithmic Language ALGOL 68. *ALGOL Bull., Sup 26*, 1–84. <https://doi.org/10.5555/1064072.1064073>
- Wiki, A. (2022). *Typing glyphs* — *APL Wiki*,. https://aplwiki.com/index.php?title=Typing_glyphs&oldid=8288
- Wikibooks. (2022). *LaTeX/Mathematics* — *Wikibooks, The Free Textbook Project*. <https://en.wikibooks.org/w/index.php?title=LaTeX/Mathematics&oldid=4044191>
- Wikipedia contributors. (2021a). *ALGOL 68* — *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=ALGOL_68&oldid=1053624353
- Wikipedia contributors. (2021b). *Källkod* — *Wikipedia, The Free Encyclopedia*. <https://sv.wikipedia.org/w/index.php?title=K%C3%A4llkod&oldid=49213732>
- Wikipedia contributors. (2021c). *Self-hosting (compilers)* — *Wikipedia, The Free Encyclopedia*. [https://en.wikipedia.org/w/index.php?title=Self-hosting_\(compilers\)&oldid=1048801699](https://en.wikipedia.org/w/index.php?title=Self-hosting_(compilers)&oldid=1048801699)
- Wikipedia contributors. (2022). *Eqn (software)* — *Wikipedia, The Free Encyclopedia*. [https://en.wikipedia.org/w/index.php?title=Eqn_\(software\)&oldid=1084921027](https://en.wikipedia.org/w/index.php?title=Eqn_(software)&oldid=1084921027)

Bilagor

Bilaga 1. Diagram av en formell grammtik för heltals aritmetik



Bilaga 2. Full lexikal EBNF specifikation av programmeringsspråket

```
funktion_nyckelord
returnera_nyckelord
om_nyckelord
medan_nyckelord
avbryt_nyckelord
fortsätt_nyckelord
variabel_nyckelord
konstant_nyckelord
inget_nyckelord
sant_nyckelord
falskt_nyckelord

kommentar          = en_rads_kommentar | fler_rads_kommentar
en_rads_kommentar   = "//" valfri_karaktär* ny_rad
fler_rads_kommentar = "/*" valfri_karaktär* "*/"

makro               = en_rads_makro | fler_rads_makro
makro_identifierare = valfri_karaktär*
makro_värde         = valfri_karaktär*
en_rads_makro       = "#(" makro_identifierare ")" makro_värde ny_rad
en_rads_makro       = "#(" makro_identifierare ")" "{" makro_värde "}"

operator            = "=" | "+" | "-" | "*"
                    | "/" | "%" | "==" | "!="
                    | "<" | "<=" | ">" | ">="
                    | "!" | "||" | "&&"

boolesk             = sant_nyckelord | falsk_nyckelord

siffror             = "0" | "1" | "2" | "3" | "4"
                    | "5" | "6" | "7" | "8" | "9"
nummer             = siffror+ "." {siffror+}

sträng_värde        = valfri_karaktär*
sträng              = "'" (sträng_värde - "\"") "'"

värde               = boolesk | nummer | sträng |

identifierare        = unicode_bokstav
                    (unicode_bokstav
                    | unicode_symbol
                    | unicode_nummer)*
```

Bilaga 3. Full syntax EBNF specifikation av programmerings-språket

```
villkors_uttryck      = om_nyckelord uttryck
                        uttryck
                        annars_nyckelord
                        uttryck

funktions_anrop       = identifierare
                        "(" uttryck
                        ("," uttryck)* ")"

gruppering           = "(" uttryck ")"

uttryck               = unärt_uttryck
                        | binärt_uttryck
                        | villkors_uttryck
                        | funktions_anrop
                        | identifierare
                        | gruppering
                        | värde

villkors_sats         = om_nyckelord
                        "(" uttryck ")" sats
                        (annars_nyckelord sats)?

returnera             = returnera_nyckelord uttryck
variable_deklaration = variabel_nyckelord
                        identifierare "=" uttryck
konstant_deklaration = konstant_nyckelord
                        identifierare "=" uttryck

kod_block             = "{" sats* "}"

sats                  = villkors_sats
                        | medan_sats
                        | returnera
                        | variable_deklaration
                        | konstant_deklaration
                        | kod_block
                        | uttryck
                        | makro

funktions_deklaration = funktion_nyckelord
                        identifierare
                        "(" identifierare
                        ("," identifierare)* ")"
                        kod_block

program               = funktions_deklaration*
```

Bilaga 4. Exempelprogram uttryck

Engelska:

```
function entry() {  
    print(1 + 2 * (3 / 4) % 8 == 2)  
}
```

Svenska:

```
funktion ingång() {  
    skriv(1 + 2 * (3 / 4) % 8 == 2)  
}
```

Tyska:

```
funktion eingang() {  
    schreiben(1 + 2 * (3 / 4) % 8 == 2)  
}
```

Franska:

```
function entrée() {  
    écrivez(1 + 2 * (3 / 4) % 8 == 2)  
}
```

Bilaga 5. Exempelprogram satser

Engelska:

```
function entry() {  
    constant y = 2  
    variable x = 0  
  
    while (x <= 10) {  
        x = x + y  
        print(x)  
  
        if (x % 2 == 0) {  
            print("a")  
        } else {  
            print("b")  
        }  
    }  
}
```

Svenska:

```
funktion ingång() {  
    konstant y = 2  
    variabel x = 0  
  
    medan (x <= 10) {  
        x = x + y  
        skriv(x)  
  
        om (x % 2 == 0) {  
            skriv("a")  
        } else {  
            skriv("b")  
        }  
    }  
}
```

Tyska:

```
funktion eingang() {  
    konstante y = 2  
    variable x = 0  
  
    während (x <= 10) {  
        x = x + y  
        schreiben(x)  
  
        ob (x % 2 == 0) {  
            schreiben("a")  
        } else {  
            schreiben("b")  
        }  
    }  
}
```

Franska:

```
function entrée() {  
  constant y = 2  
  variable x = 0  
  
  pendant (x <= 10) {  
    x = x + y  
    écrivez(x)  
  
    si (x % 2 == 0) {  
      écrivez("a")  
    } else {  
      écrivez("b")  
    }  
  }  
}
```