RUST DE TOEKOMST VAN DE OPLEIDING?

ABSTRACT

Dit onderzoek gaat over de toepasbaarheid van de taal Rust. Er wordt onderzocht of Rust een goede aanvullende taal is voor de opleiding HBO-ICT met afstudeerrichting Technische Informatica. Eerst zal er gekeken worden naar wat de taal Rust te bieden heeft later wordt er gekeken waar de taal Rust mogelijk een plek zou kunnen hebben binnen de opleiding

Mike Hilhorst
1676029

Voorwoord

Deze scriptie is geschreven voor mijn onderzoeksemester. Deze scriptie bevat een onderzoek naar wat de programmeertaal Rust kan betekenen voor de Hogeschool Utrecht. Mijn begeleider Brian van der Bijl heeft mij erg goed geholpen met dit onderzoek. Zijn vele ideeën hebben geholpen dit onderzoek te maken zoals het nu is. Deze scriptie is gemaakt door Mike Hilhorst september 2018 – januari 2019.

Managementsamenvatting

Voor het onderzoek heb ik onderzocht wat de programmeertaal Rust kan betekenen voor de Hogeschool Utrecht.

Ik heb hier voor 5 deelvragen opgesteld:

- Voor welke platformen is/ wordt Rust ontwikkeld?
- Vanuit welke talen leent Rust concepten, wat zijn deze concepten?
- Wat zijn de voor- en nadelen van Rust tegenover C++ in de scope van de lesstof van de opleiding hbo-ICT met afstudeerrichting Technische Informatica?
- Wat zou een student leren van Rust tegenover C++ scope van de opleiding hbo-ICT met afstudeerrichting Technische Informatica?
- Waar zou de taal Rust een plek hebben in de opleiding hbo-ICT met afstudeerrichting Technische Informatica?

Ik ben hier tot de volgende conclusies gekomen:

- Rust is voornamelijk voor desktopcomputers ontwikkeld, maar kan ook op veel andere platformen.
- Rust leent veel concepten uit andere talen voornamelijk C++ en SML.
- Rust is veiliger dan C++ maar minder snel en heeft slechtere documentatie. Aan de documentatie wordt momenteel hard gewerkt.
- Een student zal voornamelijk bewuster worden van geheugen-management.
- De taal Rust zou een plek kunnen hebben in het van Advanced Technical Programming

Ik ben tot deze conclusies gekomen doormiddel van een enquête, literatuurstudies en interviews. Als conclusie op mijn hoofdvraag ben ik gekomen tot:

De taal Rust kan wat betekenen voor het vak Advanced Technical Programming. Door de taal, zijn functionele eigenschappen en de C++ interface mogelijkheden, zou de taal goed aansluiten bij het vak.

Inhoudsopgave

Voorwoord 1

Managementsamenvatting 1

Inleiding 3

Aanleiding 3

Kwestie 3

Aanpassingen 3

Methodematrix 4

Over Rust 5

Beantwoording deelvragen 6

- Voor welke platformen is/wordt Rust ontwikkeld? 6
  - Niveau 1 6
  - Niveau 2 6
  - Niveau 3 7
  - Niveau 4 8
- Vanuit welke talen leent Rust concepten, wat zijn deze concepten? 9
- Wat zijn de voor- en nadelen van Rust tegenover C++ in de scope van de lesstof van de opleiding HBO-ICT met afstudeerrichting Technische Informatica? 10
  - Hoe is dit gemeten? 11
  - Hoe zijn de programma's getimed? 12
  - Hoe is het geheugen gebruik gemeten? 12
  - Hoe is de source file gemeten? 12
  - Hoe is de CPU lading gemeten? 12
  - Conclusie performance 12
  - Conclusie 14
- Wat zou een student leren van Rust tegenover C++ scope van de opleiding HBO-ICT met afstudeerrichting Technische Informatica? 15
  - Geheugen 15
  - Concurrentie 15
- Waar zou de taal Rust een plek hebben in de opleiding HBO-ICT met afstudeerrichting Technische Informatica? 16

Conclusie 17

Aanbevelingen 17

Evaluatie procesgang 17

Literatuurlijst 18

Inleiding

De hogeschool Utrecht geeft momenteel vijf programmeer cursussen op de opleiding hbo-ICT met afstudeerrichting Technische informatica, een hiervan wordt gegeven in Python en vier worden gegeven met C++. De hogeschool wil dit graag uitbreiden, ze willen in het derde en/of het vierde jaar de studenten de mogelijkheid geven om te verdiepen. Rust zou hier geschikt voor kunnen zijn omdat het op C++ lijkt maar toch een andere taal is met zijn eigen voor- en nadelen. Rust gaat veel veiliger om met geheugenmanagement en belooft dit niet terug te laten zien in de performance. Dit zou ervoor kunnen zorgen dat er grotere projecten geschreven kunnen worden, omdat er bijvoorbeeld minder tijd zou worden besteed aan memory gerelateerde bug preventie en fixing.

Aanleiding

De Hogeschool Utrecht wil graag de opleiding hbo-ICT met afstudeerrichting Technische Informatica blijven verbeteren en uitbreiden, zodat de studenten een zo goed mogelijke opleiding kunnen krijgen. Om dit te kunnen realiseren moet de Hogeschool Utrecht onderzoek doen naar het verbeteren van de opleiding. Daarom wil de Hogeschool Utrecht graag weten of nieuwe opkomende talen iets kunnen betekenen voor de opleiding, qua nieuwe kennis in beweging zetten en/of nieuwe denkwijze aanleren.

Kwestie

De Hogeschool Utrecht wil graag weten wat de (relatieve) nieuwe taal Rust kan betekenen voor de opleiding hbo-ICT met afstudeerrichting Technische Informatica. Hieruit heb ik de volgende hoofdvraag op kunnen stellen: “Wat kan de programmeertaal Rust taal betekenen voor de opleiding hbo-ICT met afstudeerrichting Technische Informatica? “.

Aanpassingen

Tijdens het onderzoek ben ik erachter gekomen dat de deelvraag: “Hoe werkt Rust intern op low- en highlevel?” niet veel nieuwe informatie zou opbrengen. Daarom zou de deelvraag ook niet veel bijdragen aan het beantwoorden van mijn hoofdvraag. Ik beschrijf in het verslag bepaalde aspecten van Rust die anders binnen deze deelvraag zouden vallen. Daarom heb ik ervoor gekozen om deze deelvraag te laten vervallen.

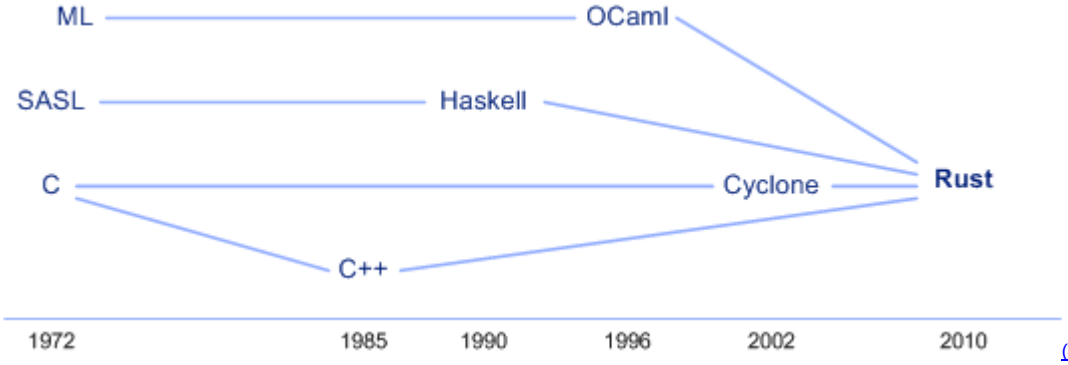
Methodematrix

#	Deelvraag	Type Deelvraag	Methode dataverzameling	Methode analyse	Resultaat
1	Voor welke platformen is/wordt Rust ontwikkeld?	Beschrijvend	Literatuurstudie	Inhoudsanalyse	Een beschrijving van de platformen waar de taal Rust voor is ontwikkeld
2	Vanuit welke talen leent Rust concepten, wat zijn deze concepten?	Beschrijvend	Literatuurstudie	Inhoudsanalyse	Een lijst van talen waaruit Rust Concepten leent en beschrijvingen van deze concepten
3	Wat zijn de voor- en nadelen van Rust tegenover C++ in de scope van de lesstof van de opleiding hbo-ICT met afstudeerrichting Technische Informatica?	Vergelijkend	Literatuurstudie en experiment	Vergelijken van data, schrijven en testen van code	Een vergelijking tussen C++ en Rust. In snelheid, geheugen gebruik en het gebruik zelf
4	Wat zou een student leren van Rust tegenover C++ scope van de opleiding hbo-ICT met afstudeerrichting Technische Informatica?	Vergelijkend en evaluerend	Literatuurstudie	Vergelijken van data	Een uitleg van wat de eventuele meerwaarde is van Rust. En wat een student leert van code schrijven in Rust i.p.v. C++
5	Waar zou de taal Rust een plek hebben in de opleiding hbo-ICT met afstudeerrichting Technische Informatica?	Evaluerend	Semigestructureerd interview	Categoriseren	Een advies over de taal en zijn plek in de opleiding

Over Rust

Uit recente enquête van Stack overflow blijkt dat bijna 80% van de gevraagde Rust met plezier gebruikt of het wil gaan gebruiken(4.0) Rust is een relatieve nieuwe taal, de taal is pas 8 jaar oud, als u kijkt naar de voorgangers van de taal zoals C ziet u dat deze taal als 46 jaar oud is. Dit is een voordeel en een nadeel. Een nadeel is dat er weinig (en/of incomplete) documentatie bestaat van de taal mede doordat veel wordt veranderd per update. Het voordeel is dat de taal van andere talen heeft geleerd.
Rust was eerst zwaar beïnvloed door Cyclone (een dialect van C). Sommige aspecten zijn uit C++ geleend bijvoorbeeld smart-pointers. Er is ook geleend van Haskell en Ocmal en nog vele andere.

Dit alles resulteert in een C-achtige taal die multi-paradigma-programmering ondersteunt.



(4.1)

Rust herkent de standaard types heel goed, dit is een beetje te vergelijken met het auto type van C++, Rust definieert zelf het type tenzij er een ander type wordt mee gegeven via argumenten. Door deze types leer je wat de daadwerkelijke grootte is van je variabelen.

```
let integer: i64 = 8; //signed 64bit integer.
```

Rust heeft 2 aparte types dit zijn: “isize” en “usize”. Deze types kunnen een per architectuur een andere waarde krijgen. Op een 32bit architectuur hebben deze een lengte van 32bit, op een 64bit systeem hebben deze types een lengte van 64bit.

```
let mut counter: usize = 0; //Dit is op een 64bit systeem, dus 64bit unsigned interger.
```

Rust ondersteunt standaard veel types.(meer over dit)

Beantwoording deelvragen

Hier worden de deelvragen beantwoordt.

Voor welke platformen is/wordt Rust ontwikkeld?

Rust ondersteunt vele platformen. Denk aan bijvoorbeeld Windows, MacOS, Linux en embedded microcontrollers. Rust ondersteunt deze platvormen op verschillende niveaus. Er zijn 4 niveaus, deze verschillen per ondersteuning, denk hierbij aan ondersteuning voor de standaard bibliotheek, Rustc en cargo.
Niveau 3 is per ongeluk ontstaan en nieuwe platformen zullen dit niveau nooit krijgen.

Niveau 1

Dit niveau is goed ondersteunt en kan gezien worden als “gegarandeerd werkend”.
Deze platformen hebben de volgende eigenschappen:
Officiële binary’s worden geleverd voor deze platformen.
Automatische testen bestaan voor deze platformen.
Veranderingen in de taal zijn master, worden pas doorgelaten nadat de testen geslaagd zijn
Documentatie over gebruik is beschikbaar voor deze platformen

Target	std	Rustc	cargo	notes
i686-apple-darwin	✓	✓	✓	32-bit OSX (10.7+, Lion+)
i686-pc-windows-gnu	✓	✓	✓	32-bit MinGW (Windows 7+)
i686-pc-windows-msvc	✓	✓	✓	32-bit MSVC (Windows 7+)
i686-unknown-linux-gnu	✓	✓	✓	32-bit Linux (2.6.18+)
x86_64-apple-darwin	✓	✓	✓	64-bit OSX (10.7+, Lion+)
x86_64-pc-windows-gnu	✓	✓	✓	64-bit MinGW (Windows 7+)
x86_64-pc-windows-msvc	✓	✓	✓	64-bit MSVC (Windows 7+)
x86_64-unknown-linux-gnu	✓	✓	✓	64-bit Linux (2.6.18+)

Niveau 2

Dit niveau is ondersteunt en kan gezien worden als “gegarandeerd bouwend”.
Deze platformen hebben de volgende eigenschappen:
Officiële binarys worden geleverd voor deze platformen.
Automatische testen zijn opgezet voor deze platformen, maar deze kunnen misschien niet gerund worden.
Veranderingen in de taal zijn master, worden goedgekeurd als het platform bouwt. Voor sommige talen houdt dit in dat alleen de standaard bibliotheek gecompileerd moet worden voor andere Rustc en Cargo ook.

Target	std	Rustc	cargo	notes
aarch64-apple-ios	✓			ARM64 iOS
aarch64-fuchsia	✓			ARM64 Fuchsia
aarch64-linux-android	✓			ARM64 Android
aarch64-unknown-linux-gnu	✓	✓	✓	ARM64 Linux
aarch64-unknown-linux-musl	✓			ARM64 Linux with MUSL
arm-linux-androideabi	✓			ARMv7 Android
arm-unknown-linux-gnueabi	✓	✓	✓	ARMv6 Linux
arm-unknown-linux-gnueabihf	✓	✓	✓	ARMv6 Linux, hardfloat
arm-unknown-linux-musleabi	✓			ARMv6 Linux with MUSL
arm-unknown-linux-musleabihf	✓			ARMv6 Linux, MUSL, hardfloat
armv5te-unknown-linux-gnueabi	✓			ARMv5TE Linux
armv7-apple-ios	✓			ARMv7 iOS, Cortex-a8

armv7-linux-androideabi	✓			ARMv7a Android
armv7-unknown-linux-gnueabihf	✓	✓	✓	ARMv7 Linux
armv7-unknown-linux-musleabihf	✓			ARMv7 Linux with MUSL
armv7s-apple-ios	✓			ARMv7 iOS, Cortex-a9
asmjs-unknown-emscripten	✓			asm.js via Emscripten
i386-apple-ios	✓			32-bit x86 iOS
i586-pc-windows-msvc	✓			32-bit Windows w/o SSE
i586-unknown-linux-gnu	✓			32-bit Linux w/o SSE
i586-unknown-linux-musl	✓			32-bit Linux w/o SSE, MUSL
i686-linux-android	✓			32-bit x86 Android
i686-unknown-freebsd	✓	✓	✓	32-bit FreeBSD
i686-unknown-linux-musl	✓			32-bit Linux with MUSL
mips-unknown-linux-gnu	✓	✓	✓	MIPS Linux
mips-unknown-linux-musl	✓			MIPS Linux with MUSL
mips64-unknown-linux-gnuabi64	✓	✓	✓	MIPS64 Linux, n64 ABI
mips64el-unknown-linux-gnuabi64	✓	✓	✓	MIPS64 (LE) Linux, n64 ABI
mipsel-unknown-linux-gnu	✓	✓	✓	MIPS (LE) Linux
mipsel-unknown-linux-musl	✓			MIPS (LE) Linux with MUSL
powerpc-unknown-linux-gnu	✓	✓	✓	PowerPC Linux
powerpc64-unknown-linux-gnu	✓	✓	✓	PPC64 Linux
powerpc64le-unknown-linux-gnu	✓	✓	✓	PPC64LE Linux
s390x-unknown-linux-gnu	✓	✓	✓	S390x Linux
sparc64-unknown-linux-gnu	✓			SPARC Linux
sparcv9-sun-solaris	✓			SPARC Solaris 10/11, illumos
wasm32-unknown-unknown	✓			WebAssembly
wasm32-unknown-emscripten	✓			WebAssembly via Emscripten
x86_64-apple-ios	✓			64-bit x86 iOS
x86_64-fuchsia	✓			64-bit Fuchsia
x86_64-linux-android	✓			64-bit x86 Android
x86_64-rumprun-netbsd	✓			64-bit NetBSD Rump Kernel
x86_64-sun-solaris	✓			64-bit Solaris 10/11, illumos
x86_64-unknown-cloudabi	✓			64-bit CloudABI
x86_64-unknown-freebsd	✓	✓	✓	64-bit FreeBSD
x86_64-unknown-linux-gnux32	✓			64-bit Linux
x86_64-unknown-linux-musl	✓			64-bit Linux with MUSL
x86_64-unknown-netbsd	✓	✓	✓	NetBSD/amd64
x86_64-unknown-redox	✓			Redox OS

Niveau 3

Dit niveau is ondersteunt en kan gezien worden als “gegarandeerd bouwend”, maar dan zonder beschikbare versies via Rustup. Er worden geen automatische testen gerund dus er wordt niet gegarandeerd dat de geproduceerd build werkt, maar meestal werken deze platformen best goed.
Deze platformen hebben de volgende eigenschappen:
Automatische testen zijn opgezet voor deze platformen, maar deze kunnen misschien niet gerund worden. Veranderingen in de taal zijn master, worden goedgekeurd als het platform bouwt. Voor sommige talen houdt dit in dat alleen de standaard bibliotheek gecompileerd moet worden voor andere Rustc en Cargo ook.

Target	std	Rustc	cargo	notes
aarch64-unknown-cloudabi	✓			ARM64 CloudABI
armv7-unknown-cloudabi-eabihf	✓			ARMv7 CloudABI, hardfloat
i686-unknown-cloudabi	✓			32-bit CloudABI
powerpc-unknown-linux-gnuspe	✓			PowerPC SPE Linux
sparc-unknown-linux-gnu	✓			32-bit SPARC Linux

Niveau 4

De Rust codebase heeft ondersteuning voor deze platformen, maar worden niet gebouwd of getest. Deze platformen kunnen misschien niet werken, verder zijn er geen officiële versies beschikbaar

Target	std	Rustc	cargo	notes
i686-pc-windows-msvc (XP)	✓			Windows XP support
i686-unknown-haiku	✓			32-bit Haiku
i686-unknown-netbsd	✓			NetBSD/i386 with SSE2
le32-unknown-nacl	✓			PNaCl sandbox
mips-unknown-linux-uclibc	✓			MIPS Linux with uClibc
mipsel-unknown-linux-uclibc	✓			MIPS (LE) Linux with uClibc
msp430-none-elf	*			16-bit MSP430 microcontrollers
sparc64-unknown-netbsd	✓	✓		NetBSD/sparc64
thumbv6m-none-eabi	*			Bare Cortex-M0, M0+, M1
thumbv7em-none-eabi	*			Bare Cortex-M4, M7
thumbv7em-none-eabihf	*			Bare Cortex-M4F, M7F, FPU, hardfloat
thumbv7m-none-eabi	*			Bare Cortex-M3
x86_64-pc-windows-msvc (XP)	✓			Windows XP support
x86_64-unknown-bitrig	✓	✓		64-bit Bitrig
x86_64-unknown-dragonfly	✓	✓		64-bit DragonFlyBSD
x86_64-unknown-haiku	✓			64-bit Haiku
x86_64-unknown-openbsd	✓	✓		64-bit OpenBSD
NVPTX	**			--emit=asm generates PTX code that runs on NVIDIA GPUs

* Dit zijn “bare-metal” microcontrollers doelen, deze hebben alleen toegang tot de kern bibliotheek en niet de standaard bibliotheek.
** Er is backend ondersteuning voor deze doelen, maar geen doelwit dat (nog) in Rustc is ingebouwd. Je moet uw eigen doelspecificatiebestand schrijven. Deze doelen ondersteunen alleen de kernbibliotheek.

Dit zijn niet de enige platforms die Rust kan compileren! Dat zijn degenen met ingebouwde doeldefinities en/of standaard bibliotheekondersteuning. Tijdens het linken van de kernbibliotheek, kan Rust zich richten op “bare metal” in de x86-, ARM-, MIPS- en PowerPC-families, hoewel hiervoor mogelijk aangepaste doelspecificaties moeten worden gedefinieerd. Al dergelijke scenario’s zijn niveau 4.

Vanuit welke talen leent Rust concepten, wat zijn deze concepten?

Rust is een relatieve nieuwe taal en leent concepten uit al bestaande talen.
Waarom leent Rust concepten uit andere talen? Omdat alles van de grond af zelf te maken is tijdsintensief en garandeert geen succes. De geleende concepten zijn bewezen goed te werken omdat ze al jaren in andere talen gebruikt worden.

De eerste release van Rust gebruikte meer concepten dan nu. Deze concepten zijn weggevallen daarom staan deze dus niet in de onderstaande lijst. In de onderstaande lijst staan de huidige nog gebruikte concepten die geleend worden. Links staat de taal waarvan een of meerder concept(en) van geleend wordt. Rechts staan de concepten die geleend worden.

Taal	Concepten
SML, Ocaml	algebraic data types, pattern matching, type inference, semicolon statement separation
C++	references, Resource Acquisition Is Initialization (RAII), smart pointers, move semantics, monomorphization, memory model
ML Kit, Cyclone	region based memory management
Haskell (GHC)	typeclasses, type families
Newsqueak, Alef, Limbo	channels, concurrency
Erlang	message passing, thread failure
Swift	optional bindings
Scheme	hygienic macros
C#	attributes
Unicode Annex #31	identifier and pattern syntax

(2)

Wat zijn de voor- en nadelen van Rust tegenover C++ in de scope van de lesstof van de opleiding HBO-ICT met afstudeerrichting Technische Informatica?

Een van de voordelen kan performance zijn. Hierdoor zouden grotere programma's uitgevoerd kunnen worden op bestaande hardware (Denk hierbij aan ARM Cortex M3 chips die de Hogeschool Utrecht gebruikt.) of bestaande programma's kunnen worden uitgevoerd op kleinere (minder snelle) hardware (denk hieraan de Atmega328 van de Arduino Nano). Dit zou geld kunnen besparen want de Arduino Due kost €35 (3.0) en de Arduino Nano kost €20. (3.1).

Korte uitleg over de data hier onder

- Source: in welke taal de code is uit gevoerd.
- Seconde: uitvoer tijd van de code.
- Geheugen: geheugen gebruikt tijdens de uitvoer (in bytes).
- Grootte: grootte van de source files (in bytes).
- CPU lading: hoeveel elke kern beladen werd tijdens de uitvoer, deze code is uitgevoerd op een quad core en daarom ook 4 percentages.

Er zijn 10 verschillende algoritmes geschreven voor 4 verschillende talen namelijk: C++, Rust, Python en Haskell.

Deze zijn uitgevoerd en dit zijn de resultaten.
De source code van deze programma's is te vinden in de appendix. (9.3)

Reverse-complement (1.0)							
Source	Seconde	Geheugen	Grootte	CPU lading			
C++	2,94	980.524	2.280	15%	50%	52%	40%
Rust	1,60	995.212	1.376	24%	25%	96%	30%
Python 3	16,76	1.005.252	814	65%	21%	44%	17%
Haskell	5,67	501.336	1.020	26%	12%	12%	40%

N-body (1.1)							
Source	Seconde	Geheugen	Grootte	CPU lading			
C++	9,42	1.712	1.763	100%	1%	2%	0%
Rust	13,25	1.808	1.805	0%	0%	1%	100%
Python 3	14min	8.212	1.196	91%	0%	1%	9%
Haskell	22,01	3.936	1.883	99%	99%	100%	99%

K-nucleotide (1.2)							
Source	Seconde	Geheugen	Grootte	CPU lading			
C++	3,83	156.104	1.624	72%	73%	98%	72%
Rust	5,98	137.956	1.648	78%	49%	90%	85%
Python 3	79,79	250.948	1.967	98%	96%	96%	99%
Haskell	35,51	604.996	1.486	89%	93%	90%	87%

Regex-redux (1.3)							
Source	Seconde	Geheugen	Grootte	CPU lading			
C++	1,83	203.680	1.315	49%	87%	57%	51%
Rust	2,44	194.804	765	85%	41%	20%	16%
Python 3	15,56	439.964	512	25%	92%	32%	32%
Haskell	Bad output						

Binary-trees				(1.4)			
Source	Seconde	Geheugen	Grootte	CPU lading			
C++	3,67	118.620	809	75%	78%	99%	76%
Rust	4,14	175.692	721	90%	90%	91%	100%
Python 3	92,72	448.844	589	87%	90%	96%	87%
Haskell	12,59	478.012	592	95%	90%	88%	94%
Fasta				(1.5)			
Source	Seconde	Geheugen	Grootte	CPU lading			
C++	1,33	1.744	2.711	82%	82%	81%	81%
Rust	1,46	3.112	1.906	84%	83%	84%	89%
Python 3	62,88	680.736	1.947	60%	56%	48%	62%
Haskell	9,46	4.980	969	100%	2%	2%	3%

Spectral-norm				(1.6)			
Source	Seconde	Geheugen	Grootte	CPU lading			
C++	1,98	1.168	1.044	100%	99%	99%	99%
Rust	1,97	2.600	1.126	100%	100%	100%	99%
Python 3	193,86	50.556	443	98%	98%	99%	99%
Haskell	4,17	3.812	987	99%	97%	99%	98%

Fannkuch-redux				(1.7)			
Source	Seconde	Geheugen	Grootte	CPU lading			
C++	10,07	1.856	980	99%	100%	100%	95%
Rust	9,87	1.848	1.020	100%	95%	100%	100%
Python 3	9 min	48.052	950	99%	100%	97%	100%
Haskell	18,20	4.024	842	99%	100%	95%	100%

Mandelbrot				(1.8)			
Source	Seconde	Geheugen	Grootte	CPU lading			
C++	1,82	29.092	1.002	98%	97%	97%	100%
Rust	1,74	33.712	1.332	98%	100%	98%	98%
Python 3	279,86	49.334	688	100%	100%	100%	100%
Haskell	11,60	38,744	782	100%	100%	100%	100%

Pidigits				(1.9)			
Source	Seconde	Geheugen	Grootte	CPU lading			
C++	1,89	4,312	513	1%	1%	99%	1%
Rust	1,74	4.520	1.366	1%	3%	0%	99%
Python 3	3,51	10.500	385	1%	1%	0%	100%
Haskell	4,20	9.724	585	27%	8%	8%	81%

Hoe is dit gemeten?

Deze testen zijn uitgevoerd op een computer met:

- Processor: Intel Q6600 quad-core op 2,4Ghz
- RAM: 4GB ram.
- OS: Ubuntu™ 18.10 Linux x64 4.18.0-10-generic.

Elk programma is uitgevoerd en gemeten bij de kleinste input waarde, de output werd opgeslagen in een bestand om vergeleken te worden met de verwachte output. Zolang de output overeen blijft komen met de verwachte output werd het programma getest met een grotere input. Totdat alle input was doorlopen. Als het programma de verwachte output bleef geven tijdens het testen (binnen een redelijke tijd) dan werd er nog 5 keer gemeten. Tijdens deze testen werd de output geledigd (/dev/null).

Als het programma niet de verwachte output leverde binnen een redelijke tijd dan werd het programma geforceerd gestopt. Als de metingen met kleinere input wel de verwachte uitvoer leverde dan werd deze 5 keer gemeten. Ook tijdens deze testen werd de output geledigd /dev/null). (1.10)

Hoe zijn de programma's getimed?

Elk programma is uitgevoerd als een "Child-process" van een Python script.
Er werd gebruik gemaakt van "time.time()" voordat het proces werd gestart en nadat deze gesloten was. (1.10)

Hoe is het geheugen gebruik gemeten?

Door "GLIBTOP_PROC_MEM_RESIDENT" te samplen voor het hoofd-proces en zijn "Child-process" elke 0,2 seconde. (1.10)

Hoe is de source file gemeten?

De code werd in een bestand gezet zonder commentaar, extra spaties en extra witregels. Op dit bestand werd een lichte GZip compressie uitgevoerd. De grootte van dit bestand werd gemeten. (1.10)

Hoe is de CPU lading gemeten?

De GTop CPU-inactief en de GTop CPU-totaal werden gemeten voor de opstarten van het "Child-process" en na het afsluiten hiervan. De percentages zijn het totaal min de inactieve waardes. (1.10)

Conclusie performance

Uitvoer snelheid in seconde

Taal	RC	NB	KN	RR	BT	FA	SN	FR	MB	PD
Rust	1,60	13,25	5,98	2,44	4,14	1,46	1,97	9,87	1,74	1,74
C++	2,94	9,42	3,83	1,83	3,67	1,33	1,98	10,07	1,82	1,89

Uit dit tabel is te halen dat Rust en C++ soms veel en soms weinig van elkaar afzitten met snelheid en dat beide talen 5 keer de snelste waren. Maar als we wat gaan rekenen dan blijkt dat als C++ sneller is, dat het dan gemiddeld ongeveer 22% sneller is dan Rust en als Rust sneller is dan is dat gemiddeld ongeveer 12%.

Geheugen gebruik in bytes

Taal	RC	NB	KN	RR	BT	FA	SN	FR	MB	PD
Rust	995.212	1.808	137.956	194.804	175.692	3.112	2.600	1.848	33.712	4.520
C++	980.524	1.712	156.104	203.680	118.620	1.744	1.168	1.856	29.092	3.212

Als we kijken naar de tabel van geheugengebruik dan blijkt dat het gebruik hoger ligt dan die van C++, met ongeveer 15%. Dat is een fors verschil, zeker voor microcontrollers. Bijvoorbeeld de Arduino Due die de Hogeschool gebruikt, deze microcontroller heeft maar 96 KB SRAM (twee banken van 64KB en 32KB) (3.0). Dit is al weinig geheugen, dus elke byte telt hier.

De source file grootte van Rust ligt gemiddeld ook boven die van C++
C++ heeft een gemiddelde grootte van 1044 bytes en Rust heeft een gemiddelde grootte 1319 bytes(1.10). Rust geeft ongeveer 26% grotere bestanden.
In het kort: Rust is gemiddeld 10% langzamer dan C++, gebruikt gemiddeld 15% meer geheugen en geeft gemiddeld 26% grotere source files.

Voordelen Rust:

- Rust is veiliger met geheugen. Rust heeft standaard veilige code, om Rust onveilig te gebruiken dan moet je een speciaal code block aanmaken met de tag ”unsafe”. C++ kan ook veilig werken, dit is optioneel, maar de gebruiker moet dit zelf toepassen. Hierdoor is het gevoelig voor fouten en fouten in geheugen-management kan leiden tot onbereikbaar geheugen of “undivided behaviour” dit houdt in dat **alles** kan gebeuren. Het is niet te bepalen wat er gebeurt, omdat je niet weet wat het stuk geheugen, dat aangesproken wordt, bevat.

- Concurrentie en threads brengen problemen met zich mee zoals: dataraces en deadlocks.
  Rust heeft ingeduwde mechanismen om dit te voorkomen. Bijvoorbeeld standaard Mutex locking tijdens communicatie tussen threads. Rust geeft threads de mogelijkheid om geheugen te lenen, dit zorgt ervoor dat bezit en de scope hiervan makkelijk en veilig door kan worden gegeven aan een andere thread.(4.1)

- Rust is jong, de taal heeft veel geleerd van zijn voorgangers en leent ook concepten. Verder ontwikkeld Rust zich snel. Het is een van de meest gelieve talen(4.0) dus er is veel interesse in de taal, wat weer goed is voor de ontwikkeling van de taal.

- Rust zijn compiler geeft aan wanneer er een fout wordt gemaakt, “Rustc” kan met 1 simpel commando een kleine uitleg geven over wat er misging. De compiler geeft ook waarschuwingen zoals *“help: remove this ‘mut’ ”,* en waarschuwt ook bij codestijl zoals hier *”variable `Example` should have a snake case name such as `example`”* dit helpt veel omdat je snel weet wat er mis gaat.

- Variabelen zijn standaard niet bewerkbaar in Rust. Als je een variabel wilt aanpassen, dan moet bij de declaratie “mut” voor de variabele zijn naam gezet worden zoals hier:
  ```
  let mut counter: usize = 0;
  ```
  Dit is een bewerkbaar variabel “counter”. Dit zorgt ervoor dat je zeer bewust bent van welke variabelen je wilt bewerken en welke niet.

Nadelen Rust:

- Door de snelle ontwikkelingen in Rust verandert er veel in een relatieve korte periode, hierdoor is documentatie matig en soms afwezig over sommigen onderwerpen. C++ is redelijk goed gedocumenteerd en heeft een grotere community, dus ontbrekende, slechte of incomplete stukken documentatie worden snel bijgewerkt.

- Embedded wordt momenteel aan gewerkt zoals te zien is in de roadmap(5.0), maar tot op heden zijn Cortex-M chips op niveau 4 ondersteunt. (De Cortex m0 en m3 chips worden gebruikt in Arduinos). Zoals eerder vermeld bij het kopje platform ondersteuning: “Deze platformen kunnen misschien niet werken, verder zijn er geen officiële versies beschikbaar.” In andere woorden er is nog niks officieels uitgebracht voor de Cortex-M chips. Embedded wordt beter ondersteunt in C++.

- De taal is relatief jong hierdoor zitten er vaker foutjes in dan in verder ontwikkelde talen. Hierdoor kan je problemen tegenkomen die je in andere talen niet zou hebben zoals C++. Bij C++ moet elke toevoeging eerst door de C++ commissie goedgekeurd worden. Hierdoor is het wel een langzamere ontwikkeling, maar wel een veiligere.

- Ik heb op Indeed gekeken naar de openstaande vacatures die naar C++ vragen en deze vergeleken met de openstaande vacatures voor Rust. Er zijn momenteel 124 openstaande vacatures voor C++ ervaren programmeurs.(6.0) Er staan maar 11 open voor Rust en deze zijn meer gefocust op web-development.(6.1)
  Op dit moment is er nog niet veel vraag naar Rust maar uit recente enquête van Stack Overflow blijkt dat bijna 80% van de gevraagde, Rust met plezier gebruikt of het wil gaan gebruiken(4.0)

- Een apart punt: Rust heeft veel andere betekenissen. Tijdens het onderzoek ben ik daar snel achter gekomen toen ik informatie zocht. Rust heeft de Nederlandse betekenis (uit)rusten, de Engelse variant Rust betekent roest en er is ook een populair spel genaamd “Rust”, dit maakt informatie zoeken tijdsintensiever, omdat er veel gefilterd moet worden.

- Rust heeft geen default parameters. Hier is een omweg voor, je kan een laag over deze functie doen. Bijvoorbeeld:

```rust
fn function (a: usize, b:usize, default_paramter: usize, default_paramter_2: usize){
    //Do stuff
}

fn function_wrapper(a: usize, b:usize){
    function(a,b,100,150);
}

fn main(){
    function_wrapper(23, 75)
}
```

  Dit gaat goed totdat de “default_parameter” wel veranderd moet worden. Dan moet er een tweede wrapper geschreven worden of moet de originele functie aangeroepen worden met de parameters ingevuld. Dit kost allemaal extra tijd.

Als je Rust en C++ vergelijkt dan valt op dat beide talen kosteloze abstractie en verplaats semantiek gebruiken. De talen hebben beide smartpointers, geen afval opruiming en veel andere overeenkomsten.

De talen verschillen wel Rust heeft bijvoorbeeld ingebouwde statische programma-analyse maar geen onbegonnen variabelen die C++ weer wel heeft.

Conclusie

In de huidige staat heeft Rust een paar voordelen over C++. Rust heeft als grootste voordeel ingebouwde veiligheid. Dit zorgt voor veiliger code. Concurrentie is best belangrijk in de huidige markt, als gewerkt wordt met gevoelige gegevens dan is het best fijn als een taal zoals Rust veiligheid afdwingt. Rust zijn automatische type herkenning is prettig omdat dit tijd scheelt. Aan de andere kant heeft C++ het auto type. C++ is beter gedocumenteerd dit geeft de taal een groot voordeel over Rust.

Als ik de talen vergelijk dan heeft Rust de volgende voordelen over C++:

- De taal is standaard veiliger.
- Tijdens het compileren wordt er aangegeven wat je fout doet.
- De taal is opgebouwd uit goede concepten uit andere talen.
- De taal is jong, dus wordt snel ontwikkeld.
- De taal heeft automatische type herkenning.

C++ over Rust:

- De taal heeft betere documentatie.
- Momenteel is embedded beter ondersteunt.
- De taal heeft minder fouten en wordt na goedkeuring van een comité aangepast.
- De markt gebruikt momenteel meer C++ dan Rust, ook al komt Rust op (4.0) C++ heeft momenteel de markt

In het kort: momenteel heeft C++ meer voordelen dan Rust.

Wat zou een student leren van Rust tegenover C++ scope van de opleiding HBO-ICT met afstudeerrichting Technische Informatica?

Geheugen

Rust geeft veel duidelijkheid met geheugen-management, als een object wordt geleend dan wordt deze standaard weggegeven, onder dit is het object niet te gebruiken, de compiler geeft een error dat deze weggegeven is en dat je geen toegang hebt.
Dit zorgt voor een veiliger en constante code. Je kan alleen geheugen adressen toespreken als ze in jouw bezit zijn. Hierdoor leer je extra bewust te zijn van je geheugen-management, want als er een fout in je code zit dan word je meteen gewaarschuwd. En de code compileert ook niet. Je moet alles in orde hebben om het (goed) te laten compileren.
De geheugenmanagent in C++ is standaard “onveilig”, hierdoor is het gevoelig voor fouten. In C++ kan je een object weggeven en daarna opvragen, je kan een geheugenadres uitlezen en/of schrijven ook al heb je het niet meer in je bezit. Dit veroorzaakt “undivided behaviour”, **alles** kan gebeuren, helaas ook wat het hoort te doen. Hierdoor is het ook moeilijk om deze fouten te vinden.

Concurrentie

Door de standaard ingebouwde veiligheid (zoals standaard Mutex locking tijdens communicatie tussen threads) is Rust zeer geschikt voor concurrentie. Met Rust zou een student minder tijd besteden aan de veiligheid van de individuele mechanismen dan bij C++. Hierdoor zou een student meer tijd kunnen besteden aan het leren van concurrent concepten.

Waar zou de taal Rust een plek hebben in de opleiding HBO-ICT met afstudeerrichting Technische Informatica?

Aangezien Rust een functionele taal is leek mij het een goede kandidaat voor het vak Advanced Technical Programming (ATP). Dit vak wordt momenteel gegeven tijdens thematieken. Om te kijken of Rust daadwerkelijk een goede kandidaat is leek het verstandig om wat opdrachten te maken van het vak ATP. Momenteel zijn deze opgegeven in Python, ik heb deze opdrachten succesvol gemaakt in Rust(9.0). Het is mogelijk Rust te gebruiken voor het vak. Toen het vak werd ontwikkeld werd de taal Haskell overwogen, maar dit is niet door gegaan omdat de taal te veel verschilde van talen die studenten hadden geleerd en de taal was moeilijk om te interfasen met de C++ gebaseerde programma's die gebruikt worden in de lessen. Rust heeft een crate "gcc" hiermee is het redelijk makkelijk om C++ functies aan te roepen(7.0). Rust zijn syntax is redelijk hetzelfde als C++. Ik als student had weinig moeite met de overstap tussen Rust en C++. Daarom lijkt Rust mij een goede kandidaat voor het vak Advanced Technical Programming.

Tijdens de interviews(9.1) die ik heb gehouden, werd het al snel duidelijk dat alle vakken in het standaard lesprogramma een bijpassende taal heeft. Het programma bestaat 3 verschillende talen: Python, assembler en C++. Python wordt een blok gegeven in het gezamenlijke deel van het eerste jaar van de opleiding, daarom is het niet verstandig om hier Rust te geven. Iedereen moet het kunnen begrijpen ook BIM'ers.

Uit mijn enquête(9.2) blijkt dat C++ makkelijker leesbaar is dan Rust. De twee blokken C++ in het tweede semester van het eerste jaar zouden vervangen kunnen worden door Rust ook al is dit moeilijker te lezen, maar door de snelle verander natuur van de taal is er momenteel weinig documentatie van Rust, dit geeft problemen tijdens het creëren van de lesstof en problemen voor student. Tijdens het Ipass project zullen ook problemen ontstaan. Hier moet een student hardware naar keuze aansturen met een Arduino Due. Hiervoor moet de student Hwlib gebruiken een C++ lib gemaakt door Wouten van Ooien(8.0). Momenteel wordt embedded matig ondersteunt door Rust en de bestaande materialen zijn al ontwikkeld.

In het tweede jaar wordt het vak concurrent system mondeling gegeven, hier wordt onder andere concurrenty aangeleerd, Rust heeft ingebouwde veiligheid voor concurrenty. In hetzelfde blok wordt lesgegeven in assembler en meer embedded C++, het project wat hieraan vast zit is embedded lasertag maken. Daarom is het moeilijk om Rust te kiezen voor dit blok.
Het blok hierna wordt C++ gegeven met de grafische bibliotheek genaamd SFML. Deze wordt niet ontwikkeld voor Rust. Het vak Algoritmes en datastructuren wordt ook gegeven tijdens dit blok. Dit vak wordt gegeven met Python, dit wordt gedaan omdat het vak meer gaat over de "algoritmes en datastructuren" en niet over de taal. Daarom is er gekozen voor een simpele taal die de student al kende.

Conclusie

Rust heeft in zijn huidige staat geen plek in het standaard lesprogramma. De taal kan misschien wel wat beteken voor het vak Advanced Technical Programming (zonder het hele lesprogramma te wijzigen).

Conclusie

Tijdens dit onderzoek heb ik veel geleerd over de taal Rust, met mijn vijf deelvragen ben ik tot vijf mini conclusies gekomen:

- Rust is voornamelijk voor desktopcomputers ontwikkeld, maar kan ook op veel andere platformen.
- Rust leent veel concepten uit andere talen voornamelijk C++ en SML.
- Rust is veiliger dan C++ maar minder snel en heeft slechter documentatie. Aan de documentatie wordt momenteel hard gewerkt.
- Een student zal voornamelijk bewuster worden van geheugen-management.
- De taal Rust zou een plek kunnen hebben in het van Advanced Technical Programming

Mijn Hoofdvraag luidt als volgt: “Wat kan de programmeertaal Rust taal betekenen voor de opleiding hbo-ICT met afstudeerrichting Technische Informatica? “.

Deze beantwoord ik als volgt:
De taal Rust kan wat beteken voor het vak Advanced Technical Programming. Door de taal zijn functionele eigenschappen en de C++ interface mogelijkheden zou de taal goed aan sluiten bij het vak.

In het huidige staat van de taal en het lesprogramma denk ik dat de taal niet veel meer kan beteken voor de opleiding hbo-ICT met afstudeerrichting Technische Informatica.

Aanbevelingen

Mijn aanbevelingen:

1. Gebruik de taal Rust in Advanced Technical Programming. Dit kan gedeeltelijk of volledig zijn. Gedeeltelijk is Rust als Python vervanger en volledig is C++ en Python vervangen met Rust. De taal lijkt op C++, dus de overstap voor studenten zal daarom niet extreem zwaar zijn.

2. Rust ontwikkelt zich momenteel zeer snel. Zoals ik beschrijf in de conclusie is de taal in zijn huidige staat niet goed aansluitend bij het huidige lesprogramma. De taal Rust kan veel gaan betekenen in de huidige markt(4.0), daarom beveel ik aan om over een tot twee jaar een vervolgonderzoek uit te voeren om te bepalen wat de programmeertaal Rust taal kan betekenen voor de opleiding hbo-ICT met afstudeerrichting Technische Informatica.

3. Als de taal verder ontwikkeld is en de documentatie bijgekomen is, dan lijkt Rust mij een goede kandidaat als gezamenlijke taal voor de opleiding hbo-ICT. De taal is krachtig maar hoeft niet zo ingewikkeld te zijn als C++ of andere C-achtige talen.

Evaluatie procesgang

Het project geen redelijk voorspoedig: ik ben 3 weken later begonnen dan normaal, dus ik had een achterstand voordat ik begon. Deze achterstand heb ik kunnen inhalen. Het project zelf ging goed. Bijna alles is gegaan zoals gepland, alleen de eindopdracht van ATP is niet helemaal gelukt. De opdracht was iets te ingewikkeld voor mijn functionele kennis en de gebruikte bibliotheken binnen Python bestaan niet voor Rust, daarom is het niet gelukt om het werkend te krijgen. Ik heb wel veel geleerd over de taal Rust, tijdens het proberen te maken van de opdrachten.

Literatuurlijst

1.0: Reverse-complement | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/revcomp.html

1.1: N-body | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/nbody.html

1.2: K-nucleotide | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/knucleotide.html

1.3 Regex-redux | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/regexredux.html

1.4: Binary-trees | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/binarytrees.html

1.5: Fasta | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/fasta.html

1.6: Spectral-norm | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/spectralnorm.html

1.7: Fannkuch-redux | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/fannkuchredux.html

1.8: Mandelbrot | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/mandelbrot.html

1.9: Pidigits | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/pidigits.html

1.10: How programs are measured | Computer Language Benchmarks Game. (z.d.). Geraadpleegd op 27 november 2018, van https://benchmarksgame-team.pages.debian.net/benchmarksgame/how-programs-are-measured.html

2.0: Appendix: Influences - The Rust Reference. (z.d.). Geraadpleegd op 27 november 2018, van https://doc.Rust-lang.org/reference/influences.html

2.1: Rust Platform Support · The Rust Programming Language. (z.d.). Geraadpleegd op 27 november 2018, van https://forge.Rust-lang.org/platform-support.html

3.0: Arduino Due. (z.d.). Geraadpleegd op 27 november 2018, van https://store.arduino.cc/arduino-due

3.1: Arduino Nano. (z.d.). Geraadpleegd op 27 november 2018, van https://store.arduino.cc/arduino-nano

4.0: Stack Overflow Developer Survey 2017. (z.d.). Geraadpleegd op 29 november 2018, van https://insights.stackoverflow.com/survey/2017?utm_source=so-owned

4.1: Why you should learn the Rust programming language. (z.d.). Geraadpleegd op 29 november 2018, van https://developer.ibm.com/articles/os-developers-know-Rust/

5.0: Rust's 2018 roadmap | Rust Blog. (2018, 12 maart). Geraadpleegd op 21 december 2018, van https://blog.Rust-lang.org/2018/03/12/roadmap.html

6.0 Rust Programmeur-vacatures - december 2018 | Indeed.nl. (z.d.). Geraadpleegd op 21 december 2018, van https://www.indeed.nl/vacatures?q=Rust%20Programmeur

6.1 C++ Programmeur-vacatures - december 2018 | Indeed.nl. (z.d.). Geraadpleegd op 21 december 2018, van https://www.indeed.nl/C++-Programmeur-vacatures

7.0 GCC-RS. (z.d.). Geraadpleegd op 21 december 2018, van https://crates.io/crates/gcc

8.0 Wouter Van Ooien hwlib | github.com (z.d.). Geraadpleegd op 21 december 2018 https://github.com/wovo/hwlib

9.0 Zie folder “ATP-opdrachten” appendix.

9.1 Zie folder “Interviews” appendix.

9.2 Zie folder “Enquête” appendix.

9.3 Zie folder “Taal test code” appendix.