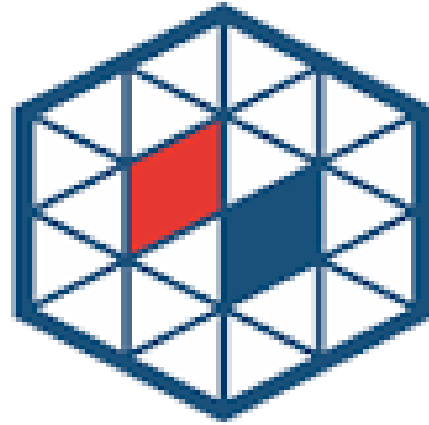


# QuickSort u AEC-u



**FERIT**

Autor:

Teo Samaržija  
(student FERIT-a)

# O AEC-u

- AEC je pojednostavljeni niski programski jezik koji sam izradio za komunikaciju sa svojim jednostavnim compilerom.
- Compiler za AEC ima oko 2000 redaka koda.

# O compileru za AEC

- Compiler za AEC pisan je u JavaScriptu i za pristupanje datotekama i druge “niže” stvari koristi Duktape radni okvir (interpreter za JavaScript pisan u C99).
- `Aec.exe` očekuje da se u direktoriju u kojem se pokrene nalaze datoteke `compiler.js` i `control.js`.

# Kako se koristi compiler za AEC?

- Compiler za AEC je jako jednostavan compiler, on ne prevodi na strojni jezik, nego samo do asemblerskog jezika.
- Ukucati u CMD: `aec <ime_programa>.aec`
- Ako je u redu, on će stvoriti datoteku `<ime_programa>.asm`.

Command Prompt

Microsoft Windows [Version 10.0.17763.805]  
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd %homedir%/Documents/aec

C:\Users\Teo Samar\Documents\AEC>aec qsort.aec

Reading JavaScript from file!

Initializing Duktape!

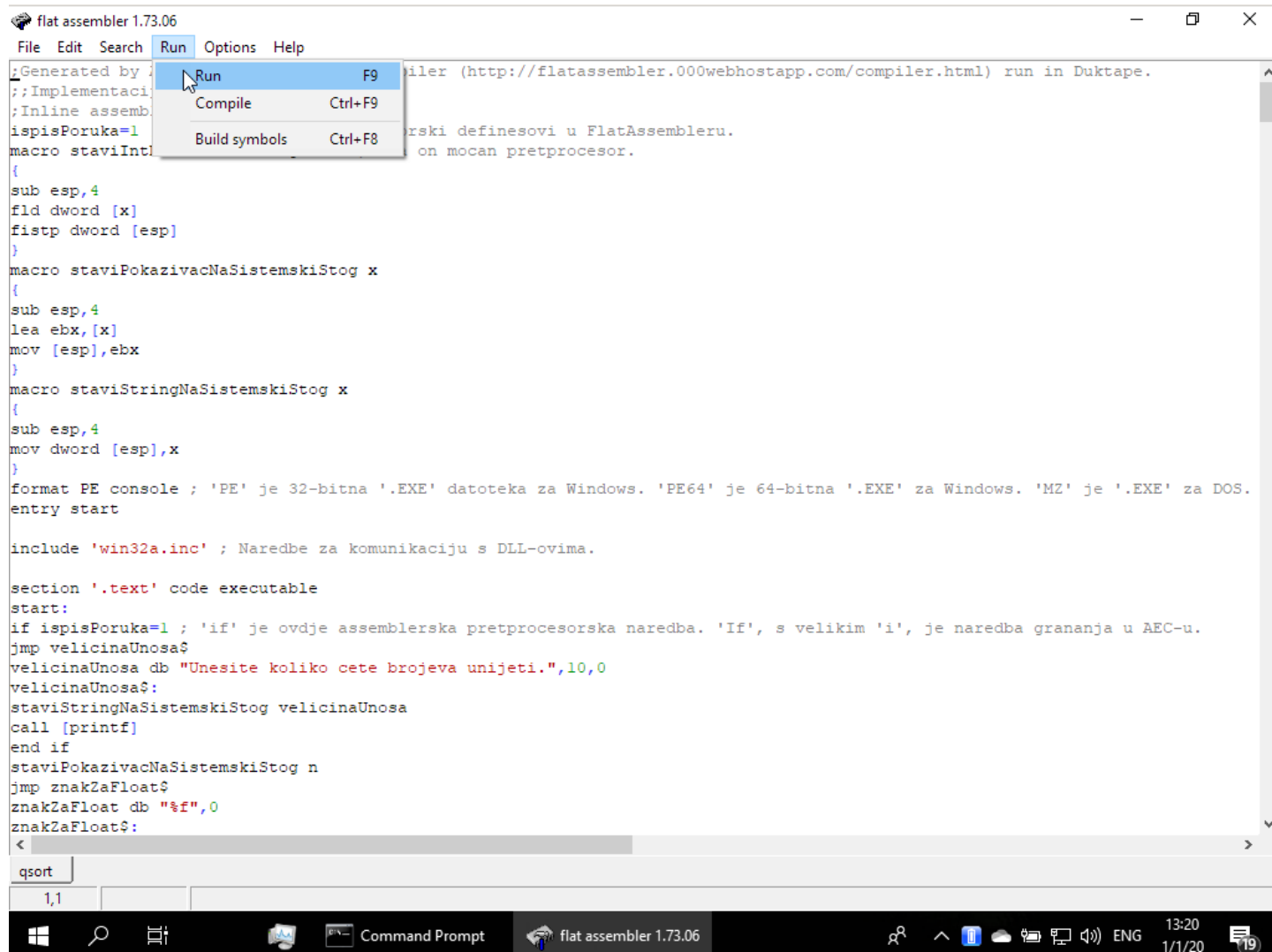
Compilation started!

Compilation finished without errors!

C:\Users\Teo Samar\Documents\AEC>qsort.asm

# Što onda s ASM datotekom?

- Treba je otvoriti u FlatAssembler IDE-u, otvoriti izbornik *Run* i stisnuti *Compile and Run*.
- Ne pokušavati kompilirati FlatAssemblerom izravno iz CMD-a, FlatAssemblerov pretprocesor će se buniti jer on očekuje da mu IDE kaže gdje su mu *includeovi*.



# O sintaksi AEC-a

- Umetanje asemblerskog koda: između `AsmStart` i `AsmEnd`. To je potrebno napraviti barem na početku programa, jer compiler za AEC očekuje da programer komunicira s asemblerskim compilerom o tome kako će biti oblikovana izvršna datoteka (i hoće li je uopće biti, da neće biti DLL ili nešto slično).
- Svaka naredba završava novim redom (kao u asemblerskom jeziku), ne moramo stavljati `;` (kao što moramo u C-u ili VHDL-u).
- Komentari se pišu između znaka `;` i kraja reda (kao u FlatAssemblerskom dijalektu asemblerskog), ne postoje višeredovni komentari.



# Operator pridruživanja

- Operator pridruživanja, ono što je u C-u znak =, a u VHDL-u  $\leq$  ili  $:=$ , u AEC-u je uvijek  $:=$ .

# Naredbe grananja

- Naredba grananja je `If` (s velikim 'i'), nakon nje u istom redu slijedi uvjet (izraz koji se vrednuje u 0 ili 1) te novi red, a grananje završava naredbom `EndIf`.
- Postoje i naredbe `Else` i `ElseIf`.

# Petlje

- Za sada postoji samo `While` petlja, ona završava s `EndWhile`.

# Operatori

- Binarni aritmetički operatori su: +, -, \* (množenje), / (dijeljenje prvog operanda s drugim) i \ (dijeljenje drugog operanda s prvim).
- Operator obrnutog dijeljenja napravio sam za SimpleCalculator. SimpleCalculator (dostupan na mom GitHub profilu) pisan je u JavaScriptu i koristi Rhino radni okvir (za JIT prevođenje JavaScripta u Javin bytecode, da se mogu upotrebljavati Javine klase i Javini radni okviri iz JavaScripta). SimpleCalculator dijeli velik dio svog koda s ovim compilerom.
- Unarni aritmetički operatori su + i - te ++. Oprez, operator ++ ponaša se drukčije nego u C-u, on je ovdje jednostavno sinonim za +1. (++) + (++) je u C-u sintaksna greška, a ovdje se vrednuje u broj 2.
- Logički operatori su & (odgovara C-ovom &&, a ne C-ovom &) i | (C-ov ||).
- Operatori usporedbe su: = (C-ov ==), < i >. Operatori <= i >= nisu implementirani jer bi oni znatno otežali tokeniziranje izraza.

5\10

Enter your notes here:

ANS:=5.0

DEL

M+

M-

MV

MS

MC

pi

7

8

9

(

+

e

4

5

6

ln

-

rad

1

2

3

exp

\*

pow

0

.

=

)

\

,

sin

cos

tan

ctg

SH

Compile Expression to Assembly

Store Result into Variable

Converted to binary IEEE754:

01000000101000000000000000000000

# Ugrađene funkcije

- Unarne funkcije: `abs`, `sin`, `cos`, `not` (logičko negiranje, C-ov `!`) `arcsin`, `arccos`, `ctg`, `arctan`, `arcctg`, `sqrt`, `ln` (prirodni logaritam), `exp`, `log` (dekadski logaritam, ne prirodni kao u C-u) i `tan`. Trigonometrijske i ciklometrijske funkcije rade sa stupnjevima, a ne s radijanima (kao u C-u).
- Binarne funkcije:
  - `atan2(y, x)` (za razliku od C-ove funkcije s istim imenom, ova daje rezultat u kutnim stupnjevima)
  - `pow(x, y)` (sintaksni šećer za `exp(ln(x) * y)`), oprez: vraća NaN ako je  $x < 0$  ili  $x = 0$ )
  - `mod(x, y)` (radi isto što i C-ova funkcija `fmod`)

# Pisanje funkcija

- Funkcije za sada nije moguće pisati u AEC-u, no moguće je program pomoću asemblerskih isječaka oblikovati u više funkcija ili procedura, te iz asemblerskih isječaka pozivati funkcije i procedure.

# Kako su te funkcije uprogramirane u compiler?

```
function facosp() // -||- arkus kosinus (po formuli  $\pi/2 - \arcsin(x)$ )
{
    asm("fstp dword [result]");
    asm("fldpi");
    asm("fld1");
    asm("fld1");
    faddp();
    fdivp();
    asm("fld dword [result]");
    fasinp();
    fsubp();
}
```



# Variable

- U AEC-u ne mogu se deklarirati varijable. AEC-ov compiler očekuje da programer to napravi u isječcima asemblerskog koda, ovisno o tome kako želi da mu izvršna datoteka bude formatirana i u kojem asemblerskom dijalektu radi. AEC-ov compiler pokušava ne raditi pretpostavke o tim stvarima. Varijable se uvijek isto zovu u asemblerskom kodu i u AEC-ovskom kodu (u C-ovskim programima linker određuje kako će se varijable zvati u asemblerskim programima umetnutim u program pisan u C-u).
- AEC za sada podržava samo jedan tip podataka, 32-bitni decimalni broj (C-ov `float`).

# Nizovi

- Postoje samo jednodimenzionalni nizovi, oni se tretiraju na isti način kao i funkcije.
- Primjerice, `pomocni(staviManje) := original(i)`.
- S lijeve strane operatora pridruživanja možemo koristiti i uglate zagrade umjesto okruglih.
- Kao i varijable, deklariraju se u asemblerskom kodu.
- Pri prevođenju nizova, compiler pretpostavlja da je moguće `pushati` i `popati` 32-bitne podatke na sistemski stog i sa sistemskog stoga, te da je moguće koristiti registar `ebx` kao pokazivač, što ponekad, ovisi kako formatiramo izvršnu datoteku u asemblerskom kodu na početku programa, nije slučaj.

# Ulaz i izlaz

- Za sada je ulaz i izlaz najlakše raditi tako da se u asemblerskim isječcima pozivaju funkcije iz C-a. To je repetitivan posao sklon greškama, no njega uvelike olakšavaju FlatAssemblerove makronaredbe.

# QuickSort u AEC-u

- Zajedno sa svim dodacima u istoj datoteci, ima 267 redaka.
- Dostupan je na mom GitHub profilu, zajedno s 32-bitnom Windowsovom izvršnom datotekom te ekvivalentnim programima pisanim u Adi i C-u, te asemblerskim kodom koji je za njega ispisao moj compiler.

<https://github.com/FlatAssembler>

C:\Users\Teo Samar\Documents\QuickSort\qsort.exe

Unesite koliko cete brojeva unijeti.

7

Unesite te brojeve:

1

-2

3

-4

5

-6

7

Sortirani niz je:

-6.000000

-4.000000

-2.000000

1.000000

3.000000

5.000000

7.000000

Unutrasnja petlja izvorsila se 12 puta.

Ocekivani broj ponavljanja te petlje, po formuli  $n \cdot \log_2(n)$ , bio bi 19.7.

Sortiranje je trajalo 0 milisekundi.

Razvrstanost pocetnog niza (s) iznosila je: 0.000000

Ocekivani broj usporedbi, po formuli:

$\exp((\ln(n) + \ln(\ln(n))) * 1.05 + (\ln(n) - \ln(\ln(n))) * 0.83 * \text{abs}(2.38854 * \text{pow}(s, 7) - 0.284258 * \text{pow}(s, 6) - 1.87104 * \text{pow}(s, 5) + 0.372637 * \text{pow}(s, 4) + 0.167242 * \text{pow}(s, 3) - 0.0884977 * \text{pow}(s, 2) + 0.315119 * s))$

bio bi:  $\exp(2.742222) = 15.521433$

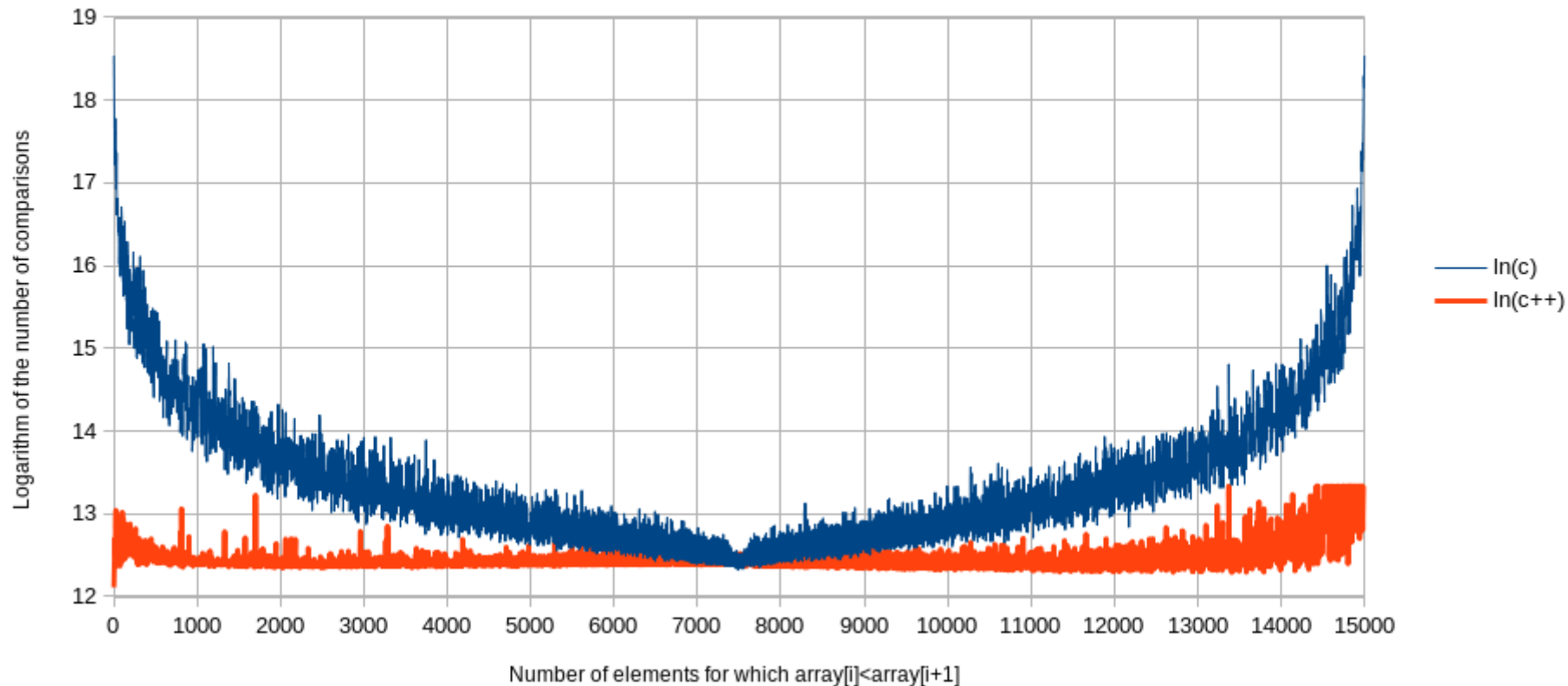
Press any key to continue . . .

# Je li moguće predvidjeti koliko će usporedbi QuickSort napraviti?

- Da bismo odgovorili na to pitanje, definirajmo za to pojam razvrstanosti niza kao broj elemenata niza za koji vrijedi da je sljedeći element niza veći od njega, podijeljeno s polovicom broja elemenata u nizu, minus jedan.
- Dakle, obrnuto poredani niz (od najvećeg prema najmanjem elementu) ima razvrstanost  $-1$ , nasumično poredani niz ima razvrstanost približno  $0$ , a poredani niz ima razvrstanost  $1$ .
- Razvrstanost se može odrediti u linearnom vremenu, višestruko brže no što bi se QuickSort vrtio i u najboljem slučaju.

Results of the test about the sortedness of an array - logarithmic scale - array is shuffled two times

How many comparisons does QuickSort do on partially sorted arrays?



Plava krivulja predstavlja moju implementaciju QuickSorta u C-u, a crvena C++-ovu naredbu `sort`.

# Formula?

- Naivno implementirani genetski algoritam, kojem su bili dostupni podaci s tog grafikona, tvrdi da je formula koja dobro opisuje broj usporedbi koje će QuickSort napraviti:

$$f(n, s) = e^{(\ln(n) + \ln(\ln(n))) \cdot 1.05 + (\ln(n) - \ln(\ln(n))) \cdot 0.83} \cdot |2.38854 \cdot s^7 - 0.284258 \cdot s^6 - 1.87104 \cdot s^5 + 0.372637 \cdot s^4 + 0.167242 \cdot s^3 - 0.0884977 \cdot s^2 + 0.315119 \cdot s|$$

Ta formula dobro opisuje kako se taj moj program ponaša za velike nizove, no ona, na žalost, znatno precjenjuje koliko će usporedbi napraviti za manje.



Hvala na pozornosti!