



**Universiteti i Prishtinës “Hasan Prishtina”**  
**Fakulteti i Inxhinierisë Elektrike dhe Kompjuterike**

**Lënda:** Arkitektura dhe organizimi i kompjutereve

**Detyra:** Seria Fibbonaci

Profesori:  
Prof. Dr. Valon Raca

Studenti:  
Enes Hasani  
ID : 190714100148

# Hyrje

## Detyra:

```
#include <iostream>

using namespace std;

int fib(int x) {
    if ((x == 1) || (x == 0)) {
        return(x);
    }
    else {
        return(fib(x - 1) + fib(x - 2));
    }
}

int main() {
    int x, i = 0;

    cout << "Enter the number of terms of series : ";

    cin >> x;

    cout << "\nFibonnaci Series : ";

    while (i < x) {
        cout << " " << fib(i);

        i++;
    }

    return 0;
}
```

## Komenti:

Ky program bënë llogaritjen dhe printimin e anëtarëve të serisë fibonacci.

Si cdo program ne c++ edhe ketu fillon ekzekutimi programit prej funksionit kryesor main. Fillimisht kemi dy variabla te deklaruar te tipit integer(numer i plote) x-in dhe i-ne. Pyesim shfrytezuesin se sa anetare te serie se deshiron t'i shfaq permes komandes cout, pastaj kete numer e ruajme ne variablen x(kjo eshte arsyeja e perdorimit te variables x). Pasi qe kemi marre numrin e anetareve qe shfrytezuesi deshiron t'i paraqes tani na mbetet ti paraqesim ata anetare(cout-i i ardhshem). Tani hyn ne perdorim variabla i. Kete variabel e perdorim si parametër për funksionin fibonacci, ku i paraqet indeksin e anetarit i te serie fibonacci( $i=0,1,2,3,\dots,x-1$ ). Ne e thirrjmë funksionin fibonacci per secilin anëtarë të vargut fibonacci(për ta gjetur vlerën atij anëtarit), gjatësia x e te cilit është përcaktuar nga shfrytëzuesi. Për këtë e perdorim një unazë while, e cila perfundon kur janë printuar të gjithë anëtarët e vargut nga anëtarit 0 deri te anëtarit x-1, andaj i-ne e inkrementojmë për 1 cdo-herë pasi të printojmë një vlerë, në mënyrë që të shkojmë te anëtarit i ardhshëm, perderisa ka ende anëtarë( $i < x$ ). Në brendi të unazës printojmë një hapësirë(me qene dalja e lexueshme), dhe e kërkojmë vlerën e anëtarit të i-te te vargut fibonacci permes funksionit fib duke ia pasuar indeksin e anëtarit në fjalë, i-ne(fib(i)).

Funksioni fib është funksion i cili si parameter formal pranon një vlerë të tipit int(indeksin e ndonjë anëtarit), po ashtu si vlerë kthyesë ka një variabel të tipit int, përkatësisht vlerën e anëtarit që i korrespondon atij indeksi që është marrë si parametër. Funksioni fib është funksion rekursiv, i cili kthen 0 kur indeksi që i pasohet i korrespondon anëtarit 0, 1 kur indeksi që i pasohet i korrespondon anëtarit 1 (base case i cili trajtohet përmes if), pastaj secili anëtarë i radhës gjenerohet si shumë e dy anëtarëve paraprak(kjo situatë trajtohet brenda else), kjo paraqet definimin e serie fibonacci, pra prej anëtarit të 2 te e tutje anëtarët mund të gjenerohen në mënyrë rekursive, ku funksioni e thirr vetveten deri sa të arrij deri tek basecase.

# Realizimi i kodit ne MIPS

Kjo detyre ka qene mjaftë sfiduese, sepse ka pasur nested-procedures, dhe jo vetem kaq ato kanë qenë rekurzive. Kjo e ka bërë të obligueshme përdorimin e stack-ut, në mënyrë që kur të mbishkruhen regjistrat të mos na humbin të dhënat, ku lind vështirsia e menaxhimit të tij dhe e përcaktimit të variablave të cilat duhet t'i ruajmë në të. Problem me theks të veçantë që kam hasur ka qenë zgjedhja e një strukture të përshtashme për vizualizimin e procesit rekurziv kur vlerat fillojnë të rriten, me ç'rast kam kuptuar se pema binare është një mjet i mirë vizualizimi për problemin në fjalë.

**Shënim:** Shpjegimi më i detajuar është bërë me komente në fajllin ku gjendet kodi. Këtu po paraqes disa pjesë kryesore.

```
.text
```

```
.globl main
```

```
fib:
```

```
addi $sp,$sp,-12
```

```
sw $ra, 8($sp)
```

```
sw $a0, 4($sp)
```

```
bne $a0,1,xZero
```

```
add $v0,$a0,$zero
```

```
addi $sp,$sp,12
```

```
jr $ra
```

```
xZero:
```

```
bne $a0,0,Else
```

```
add $v0,$a0,$zero
```

```
addi $sp,$sp,12
```

```
jr $ra
```

```
funkcioni fib sahere qe thirret  
alokon stack-un qe te ruaj  
3 te dhena(nga 4 bajt gjithsej 12 bajt  
te cilat duhet te memorizohen sepse  
perndryshe mbishkruhen neper regjistra),  
adresen ku duhet te vazhdoi kodi(ra)  
pas ekzekutimit te njerit prej blloqeve  
degezuese, parametrin x sepse regjistri  
qe e ruan mbishkruhet kur thirret funksioni  
fib per x-1 , e neve na duhet edhe per x-2  
si dhe rezultatin qe kthen fib per x-1  
sepse nese se ruajme na mbishkruhet nga fib(x-2).  
Fillimisht kontrollojme a eshte parametri x i  
barabarte me 1, nese po e kthejme 1, zbrazim stack-un  
shkojme te rreshti i kodit adresa e te cilit gjendet  
ne regjistrin qe ruan regjitri i adresave, nese jo kontrollojme  
a eshte 0(xZero),nese po kthejme 0,zbrazim stackun, shkojme te rreshti  
i kodit adresa e te cilit gjendet ne regjistrin  
qe ruan regjitri i adresave, nese nuk eshte 0 shkojme te pjesa  
ku e bejme thirrjen rekurzive(Else)
```

Else:

addi \$a0, \$a0, -1

jal fib

sw \$v0, 0(\$sp)

lw \$a0, 4(\$sp)

addi \$a0, \$a0, -2

jal fib

lw \$t0, 0(\$sp)

add \$v0, \$t0, \$v0

lw \$ra, 8(\$sp)

addi \$sp, \$sp, 12

jr \$ra

main:

addi \$s0, \$zero, 0

la \$t1, numberOfSeries

li \$v0, 4

add \$a0, \$t1, \$zero

syscall

li \$v0, 5

syscall

move \$t2, \$v0

la \$t1, fibS

li \$v0, 4

add \$a0, \$t1, \$zero

syscall

ne kete pjese zvogelojme x per 1 ne x-1, dhe duke e thirrur funksionin fib e gjejme vleren e anetarit x-1, qe te mos na mbishkruhet kur e therrasim per x-2 e ruajme ne stack kete rezultat, pastaj e gjejme vleren e anetarit x-2, pra jemi gati per ta gjetur anetarin x si shume e x-1 dhe x-2, para se t'i mbledhim duhet ta marrim nga stack vleren e anetarit x-1(lw) por duke pasur kujdes e duke e ruajtur ne nje regjister tjeter e jo ne regjistrin ku e kemi vleren e anetarit x-2, pastaj vetem i mbledhim, nga stack marrim adresen qe tregon se te cili rresht duhet te vazhdoi kodi, e zbrazim stack-un dhe shkojme te rreshti adresen e te cilit e morem nga stacku

funksioni main ku ruajme ne regjister variablen(i) qe perdoret per indeksim te anetareve, printimi i stringut numberOfSeries ku pyesim shfryetezusin per numrin e termave. Ruajtja e inputit int ne nje regjister, dhe se fundi printimi i stringut fibs ku ne vazhdim i printojme anetaret permes unazes

---

loop:

```
slt $t1,$s0,$t2
beq $t1,$zero,exit
la $t3,wSpace
li $v0,4
add $a0,$t3,$zero
syscall
move $a0,$s0
jal fib
sw $v0,factResult
li $v0,1
lw $a0,factResult
syscall
addi $a0,$a0,1      #inkrementimi per 1 i regjistrat qe ruan variablen i
j loop
```

exit:

```
li $v0,10
syscall
```

.data

numberOfSeries: .asciiz "Enter the number of terms of series : "

fibS: .asciiz "\nFibonnaci Series : "

wSpace: .asciiz " "

factResult: .word 0

```
unaza while(ketu e emeruar loop)
ne te cilen
therrasim funksionin fib
per te llogaritur vleren
e secilit anetare(factResult),
pasi qe te kthehet ajo vlere
printohet, printojme
nga nje hapesire mes tyre(wSpace)
permes syscall dhe e cila perfundon
kur jane printuar te gjitha
anetaret e kerkuar(kur x=i)
```

```
ruajtja e te dhenave(stringjeve) te
cilat nuk i zene regjistrat
ne memorie dhe e nje integer-i
te cilin do ta mbishkruajme per te
ruajtur rezultatin e funksionit fib
```

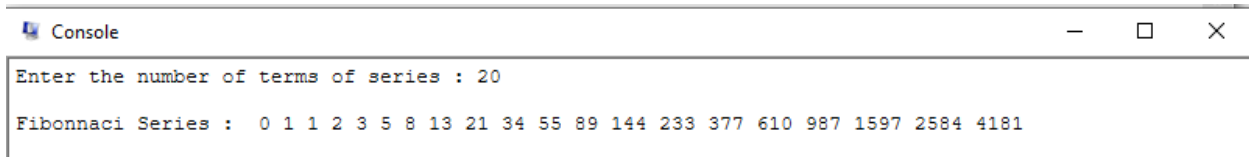
# Testimet me QtSpim

Testimet me QtSpim kanë qenë një ndihmesë e madhe e madhe në vizualizimin e ecurisë së rekurzionit, po ashtu për debugim.



```
Console
Enter the number of terms of series : 10
Fibonnaci Series : 0 1 1 2 3 5 8 13 21 34
```

Nga rezultati shihet se programi punon në rregull për 10 anëtarët e parë të serisë fibonacci, dy anëtarët e parë janë 0 dhe 1, shuma e tyre është 1 anëtar i 3 të, pastaj shuma e anëtarit të 2-të me anëtarin e 3-të na jep anëtarin e 4-të ( $1+1=2$ ), e kështu me radhë.



```
Console
Enter the number of terms of series : 20
Fibonnaci Series : 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```

Nga rezultati i mësipërm shihet se po funksionon në rregull edhe për 20 anëtarë. Një gjë që vërehet është që sa më i madh të jetë termi aq më shumë kohë merr për tu procesuar dhe mjaftë shpejt arrijmë në një pikë që llogaritja e anëtarëve bëhet në kohë jo efikente, gjë që ka kuptim plotësisht sepse koha që na duhet për të llogaritur vlerën e anëtarit të  $n$ -të është sa koha e nevojshme për llogaritjen e vlerës së anëtarit  $n-1$  plus koha e nevojshme për llogaritjen e anëtarit  $n-2$  plus koha që duhet për t'i mbledhur që mund të neglizhohet për shkak se është tejet e shkurtë.



```
C:\WINDOWS\system32\cmd.exe
Enter the number of terms of series : 20
Fibonnaci Series : 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
Press any key to continue . . .
```

I njëjti rezultat edhe nga ekzekutimi i kodit në C++, që dëshmon se kodi në assembler bën zgjidhjen e problemit në fjalë.