# monolithicFibonacci.c

```
int count = 0;
int next = 0;
int first = 0;
int second = 1;
int i = 2;

scanf("%d", &count);

if (count >= 1)
    printf("\n%d\n", first);
else
    goto end;

if (count >= 2)
    printf("%d\n", second);
else
    goto end;

redo: if(count > i){
    next = first + second;
    printf("%d\n", next);
    first = second;
    second = next;
    i++;
    goto redo;
}

    end: return 0;
```

## monolithicFibonacci.c => Monolithic Algorithm

R1: Do C = 0 goto R2;  = A
R2: Do N = 0 goto R3; = B
R3: Do F = 0 goto R4; = C
R4: Do S = 1 goto R5; = D
R5: Do I = 2 goto R6; = E
R6: Do Read(C) goto R7; = F
R7: If (C >= 1) goto R8 else goto Rx;
R8: Do Write(F) goto R9; = H
R9: If (C >= 2) goto R10 else goto Rx;
R10: Do Write(S) goto R11; = K
R11: If (C > I) goto R12 else goto Rx;
R12: Do N = F + S goto R13; = M
R13: Do Write(N) goto R14; = M
R14: Do F = S goto R15; = M
R15: Do S = N goto R16; = M
R16: Do I = I + 1 goto R11; = M

## Monolithic Algorithm => Trace Machine

*n = 4*

(0,  e)
(1,  A)
(2,  AB)
(3,  ABC)
(4,  ABCD)
(5,  ABCDE)
(6,  ABCDEF)
(7, ABCDEF)
(8, ABCDEFH)
(9, ABCDEFH)

(10, ABCDEFHK)
(11, ABCDEFHK)
        (12, ABCDEFHKM)
(11, ABCDEFHKM)
        (12, ABCDEFHKMM)
(11, ABCDEFHKMM)
(Rx, ABCDEFHKMM)

## iterativeFibonacci.c

```
int count = 0;
int next = 0;
int first = 0;
int second = 1;
int i = 2;

scanf("%d", &count);

if (count >= 1)
    printf("\n%d\n", first);

if (count >= 2)
    printf("%d\n", second);

while(count > i){
    next = first + second;
    printf("%d\n", next);
    first = second;
    second = next;
    i++;
}
```

```
return 0;
```

## iterativeFibonacci.c => Monolithic Algorithm

R1: Do C = 0 goto R2;  = A
R2: Do N = 0 goto R3; = B
R3: Do F = 0 goto R4; = C
R4: Do S = 1 goto R5; = D
R5: Do I = 2 goto R6; = E
R6: Do Read(C) goto R7; = F
R7: If (C >= 1) goto R8 else goto Rx;
R8: Do Write(F) goto R9; = H
R9: If (C >= 2) goto R10 else goto Rx;
R10: Do Write(S) goto R11; = K
R11: If (C > I) goto R12 else goto Rx;
R12: Do N = F + S goto R13; = M
R13: Do Write(N) goto R14; = M
R14: Do F = S goto R15; = M
R15: Do S = N goto R16; = M
R16: Do I = I + 1 goto R11; = M

## Monolithic Algorithm => Trace Machine

*n = 4*

(0,  e)
(1,  A)
(2,  AB)
(3,  ABC)
(4,  ABCD)
(5,  ABCDE)
(6,  ABCDEF)
(7, ABCDEF)

(8, ABCDEFH)
(9, ABCDEFH)
(10, ABCDEFHK)
(11, ABCDEFHK)
        (12, ABCDEFHKM)
(11, ABCDEFHKM)
        (12, ABCDEFHKMM)
(11, ABCDEFHKMM)
(Rx, ABCDEFHKMM)

## recursiveFibonacci.c

```
int fibonacci(int n)    F
{
    if (n <= 1)
        return n;     H
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n = 0;    A
    int i = 0;    B

    scanf("%d", &n);    C

    while (n > i)
    {
        printf("%d\n", fibonacci(i));    F
        i++;    K
    }
```

```
    return 0;
}
```

## recursiveFibonacci.c => Trace Machine

*n = 4*

(0, e)
(1, A)
(2, AB)
(3, ABC)
(4, ABC) *i = 0*
(5, ABCF)
(6, ABCF)
(7, ABCFH)
(8, ABCFHK)
(4, ABCFHK) *i = 1*
(5, ABCFHKF)
(6, ABCFHKF)
(7, ABCFHKFH)
(8, ABCFHKFHK)
(4, ABCFHKFHK) *i = 2*
(5, ABCFHKFHKF)
(6, ABCFHKFHKF)
(5, ABCFHKFHKFF)
(6, ABCFHKFHKFF)
(7, ABCFHKFHKFFH)
(5, ABCFHKFHKFFHF)
(6, ABCFHKFHKFFHF)
(7, ABCFHKFHKFFHFH)
(8, ABCFHKFHKFFHFHK)
(4, ABCFHKFHKFFHFHK) *i = 3*
(5, ABCFHKFHKFFHFHKF)

(6, ABCFHKFHKFFHFHKF)
(5, ABCFHKFHKFFHFHKFF)
(6, ABCFHKFHKFFHFHKFF)
(5, ABCFHKFHKFFHFHKFFF)
(6, ABCFHKFHKFFHFHKFFF)
(7, ABCFHKFHKFFHFHKFFFH)
(5, ABCFHKFHKFFHFHKFFFHF)
(6, ABCFHKFHKFFHFHKFFFHF)
(7, ABCFHKFHKFFHFHKFFFHFH)
(5, ABCFHKFHKFFHFHKFFFHFHF)
(6, ABCFHKFHKFFHFHKFFFHFHF)
(7, ABCFHKFHKFFHFHKFFFHFHFH)
(8, ABCFHKFHKFFHFHKFFFHFHFHK)
(Rx, ABCFHKFHKFFHFHKFFFHFHFHK)

## Comparison of Tracing Machines

M (Rx, ABCDEFGHJKLMLML)

I (Rx, ABCDEFGHJKLMLML)

R (Rx,
ABCDFGHKDFGHKDFGFGHFGHKDFGFGFGHFGHFGHKD)

## Conclusion

M = I

M ≠ R

I ≠ R