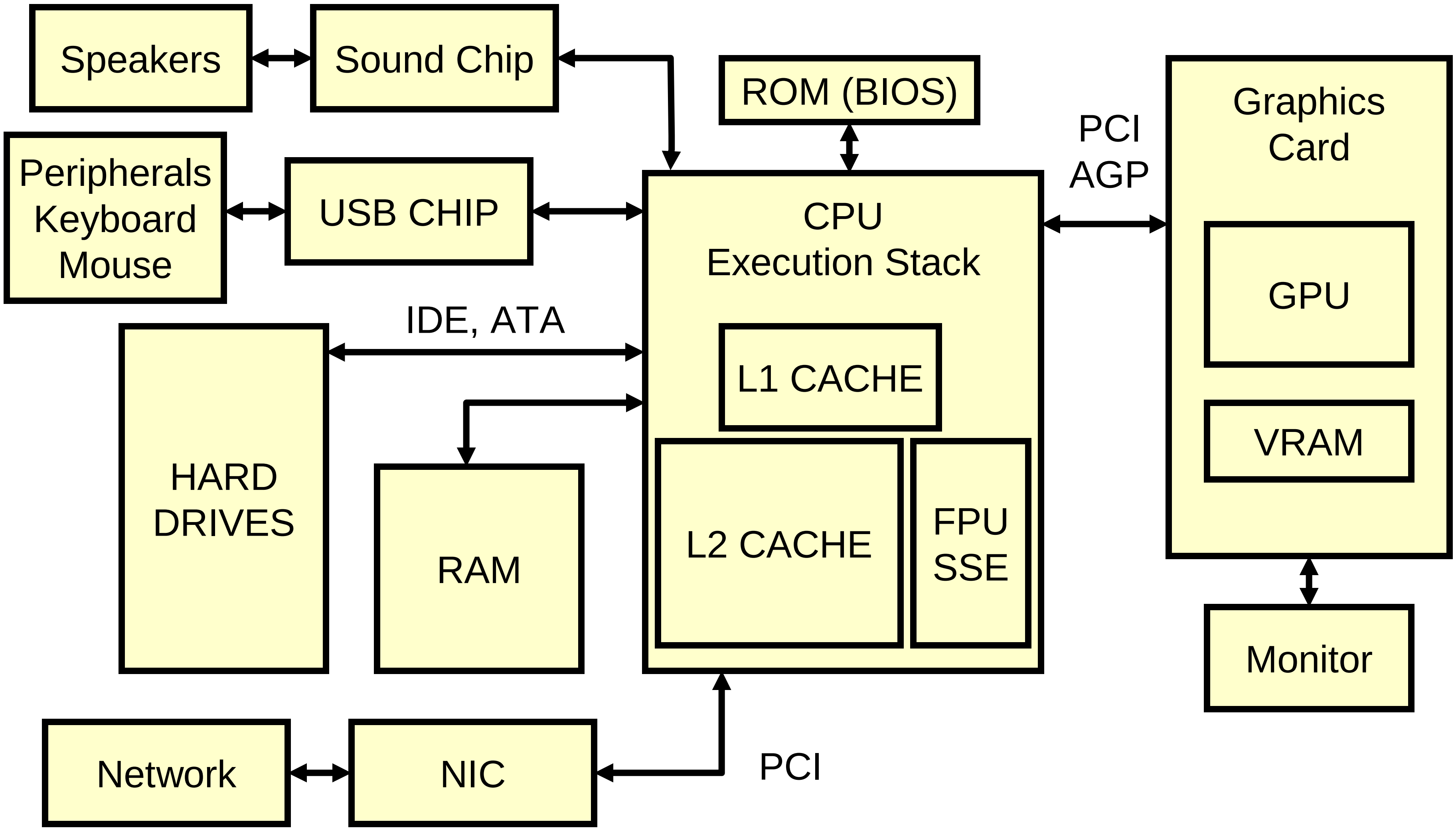
Basics of Computing

Ed Gatzke, gatzke@sc.edu

- Computers do exactly what you tell them to do
  - Garbage In / Garbage Out
- Digital information stored as 0s or 1s, low or high voltage
- Central Processing Unit (CPU)
  - Processes lists of simple instructions:
    - Move data from one memory location to another
    - Compare two pieces of memory (logic gates)
    - Specialized (Math, Graphics, Sound)
- Input / Output
  - Input: Mouse, Keyboard, File, URL/Online
  - Output: Video, Sound, File, Printouts

Hardware Schematic



Computing Hardware

- Differences in hardware
  - PC CPU not the same as (old) Mac CPU or Unix box
  - x86 (Intel or AMD) vs. PowerPC vs. MIPS vs. Alpha
- Differences in operating system
  - Windows 95, ME, XP, Vista (usually compatible)
  - Apple OSX
  - Linux, BSD, Unix
- OS is low level program that controls how programs execute and how components talk
  - Hard Drives, Memory, Keyboard, Mouse, Screen
  - Binary executable format (.exe on XP) files
  - .exe generally won't work on other platforms (exceptions)

Programs

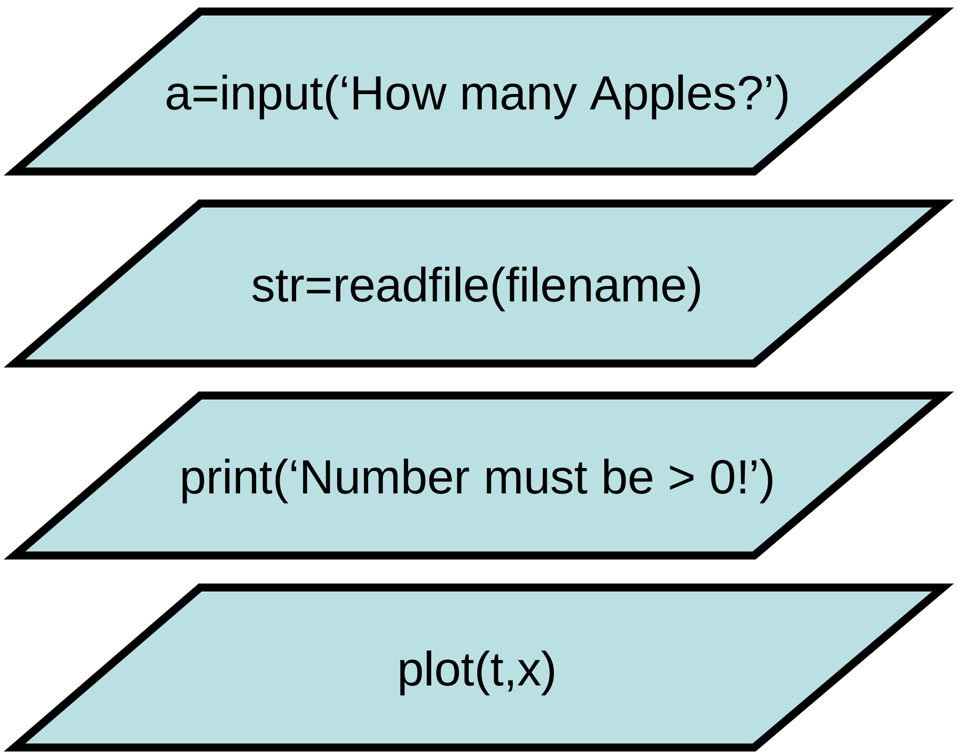
- Executable programs are build from “source code”
  - Excel, Word, Powerpoint, Explorer, Firefox
- Source code is made up of simple building blocks
  - Data types (Integers, characters, real numbers, arrays)
  - Statements (k=k+1, k=max(x), k=a*b)
  - Relational operators (<, >, ==)
  - Logical Operators (AND, OR)
  - Conditional Statements (IF condition THEN do)
  - Loops (FOR i=1:10 do, WHILE eps>0.1 do)
- Specific syntax differs from language to language
- Variable names relate to something in memory

Compiled vs. Interpreted

- Compiled languages (C, C++, Fortran)
  - Convert source code to executable format
  - Resulting program runs very quickly
  - Executable only runs on target OS + hardware
- Interpreted languages (Java, C#, Matlab, MathCad)
  - Interpreter compiled for specific OS + hardware
  - File is read by interpreter in protected “sandbox”
  - Java, C# partial compilation, bytecode
  - Matlab, can sometimes compile to executable

Flow Charting of Algorithms

- Algorithm is a method to solve a problem
- Flow charts are graphical representations of algorithms
- Data Input and Output using simple functions



Statements

- Simple execution statements in rectangles

i=i+1

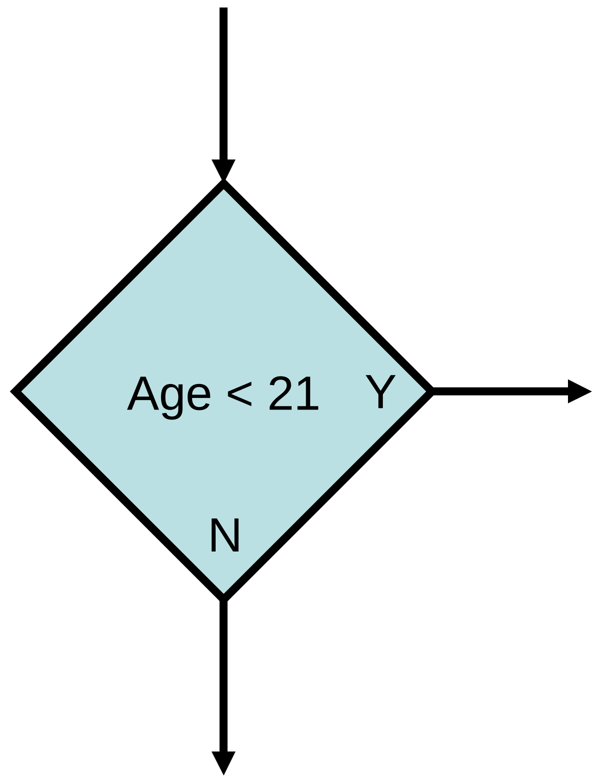
sum=a+b

a=max(x)

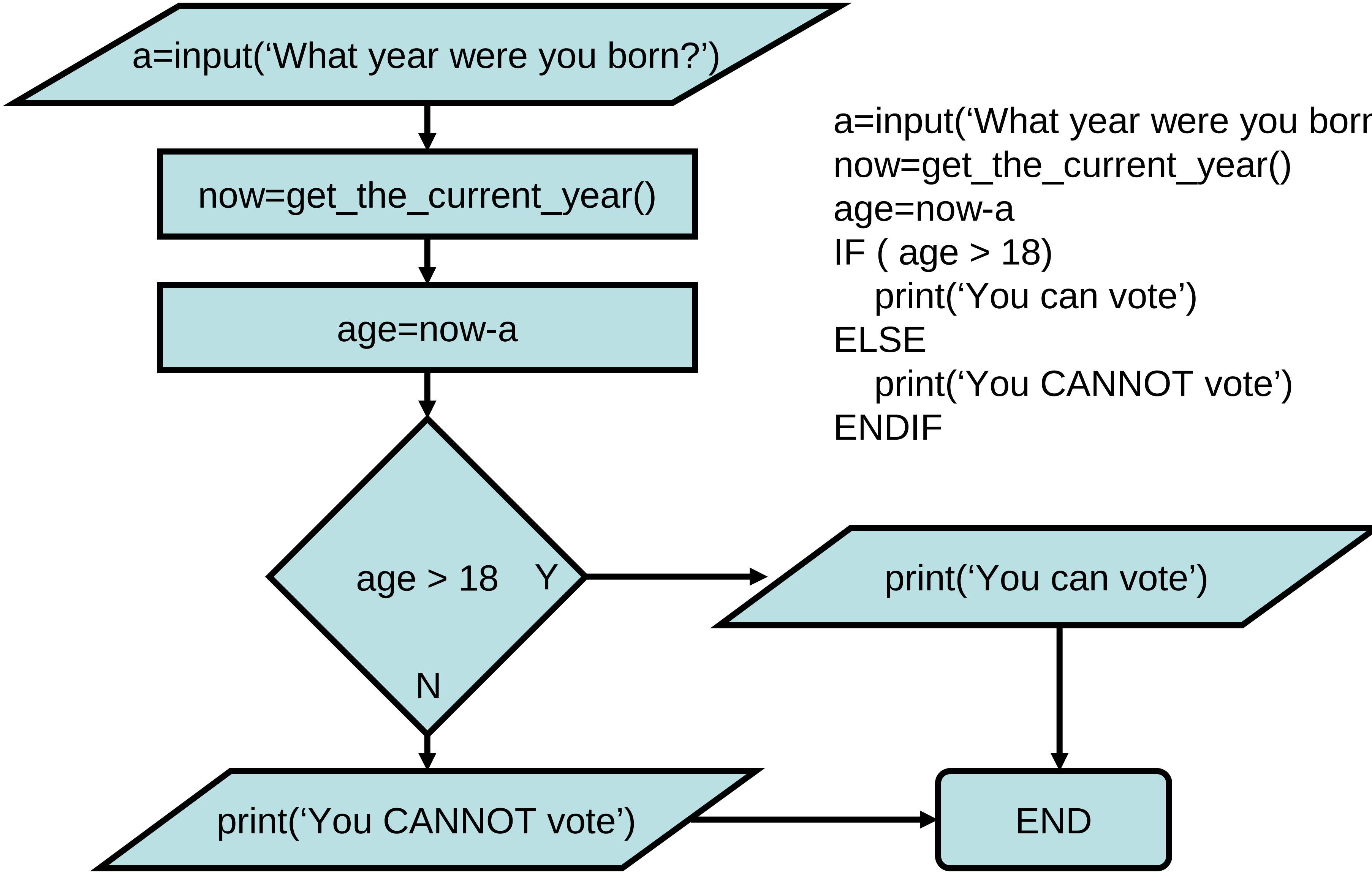
c=myfunction(a,b)

Conditional Statements

- IF THEN ELSE statements
- Branches execution flow



Example




```
a=input('What year were you born?')
now=get_the_current_year()
age=now-a
IF ( age > 18)
    print('You can vote')
ELSE
    print('You CANNOT vote')
ENDIF
```

Variable Trace

- Variables must get some initial value
- Determine values as the program executes

	a	now	age
Initial variable values:	null	null	null
a=input('What year were you born?'	1978	null	null
now=get_the_current_year()	1978	2006	null
age=now-a	1978	2006	28
IF (age > 18)	1978	2006	28

```
a=input('What year were you born?'
now=get_the_current_year()
age=now-a
IF (age > 18)
    print('You can vote')
ELSE
    print('You CANNOT vote')
END
```

Pseudo Code

- Don’t worry about syntax or details
- Conplex general method leaving out specifics

```
Get input data (may involve many steps, reading files, etc)
Check input data
IF ( data is consistent)
      Compute result
      Display result
ELSE
      Report Error and Stop
END
```

Data Types

- Boolean: TRUE or FALSE
  - Sometimes 1=true, 0=false
- Integers: -2, -1, 0, 1, 2, etc.
  - Usually limited to some range
- Floating Point: 1.43e2 => 143   -4.5e-2 => -0.045
  - sign, exponent (limited), mantissa (limited accuracy)
- Characters: ‘x’ ‘2’ ‘@’ ‘+’
- Arrays: a=[2 3 -1 2 3]
  - Vector of primitives
- Objects: complex data types
  - Person.firstname = ‘George’ <- String of Characters
  - Person.age = 22 <- integer

Statements

- Usually, each language has a set of base functions
  - min, max
  - floor round down to nearest int)
  - ceil round up to nearest int)
  - mod(x,y) remainder of x/y
  - pow(x,y) find x^y
  - sin(x) exp(x)
  - length(x) length of a vector, not always used
- You can usually write your own specialized functions
  - out=average(xvector)

Conditional Statements

- Check some condition, if met, do something

```
IF (x<y)
   print (‘x is less than y’)
ELSE
   print (‘x is NOT less than y’)
END
```

- PEMDAS Order of Operations

```
IF ( (x-1 < 0) AND (x*y<0) )
```

Use parens liberally in many cases to clarify

```
IF ( (  (x-1) < 0) AND ( (x*y) < 0) )
```

FOR Loops

- FOR: if you know how many times it should execute

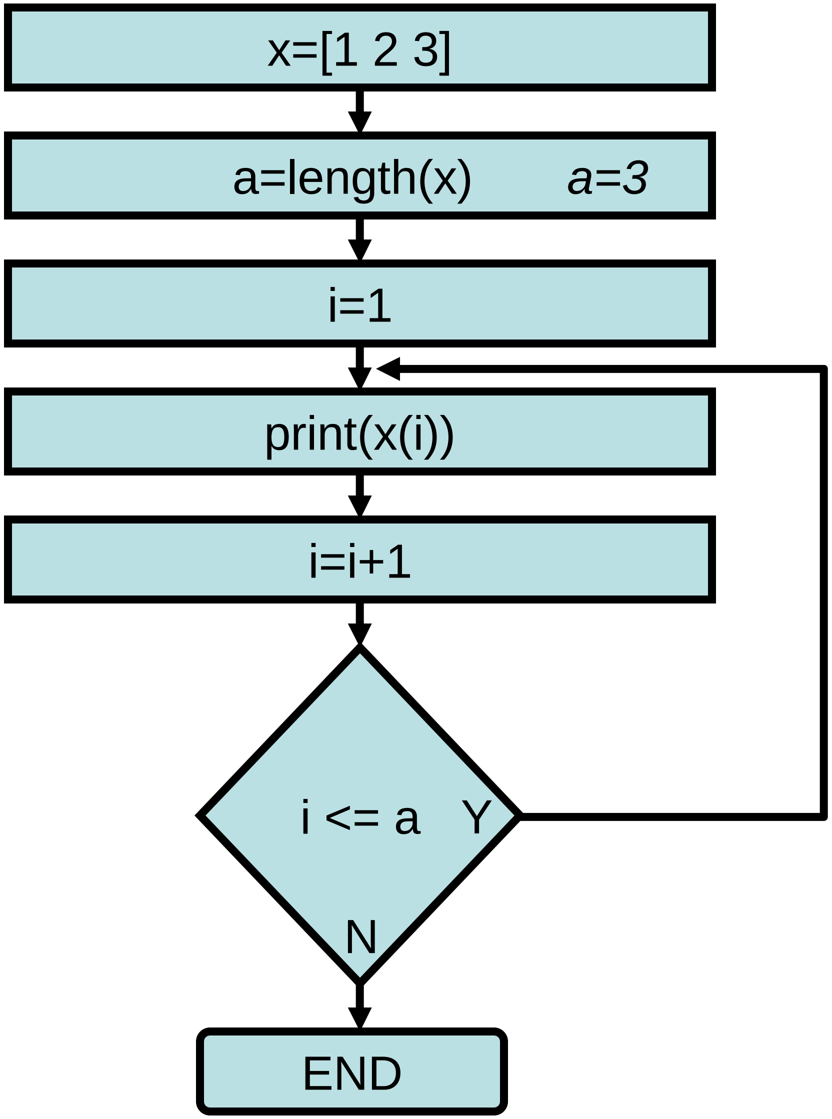
```
x=[1 2 3]
a=length(x)              <= a now is 3, not null
FOR i=1:a                <= i is 1, then 2, then 3.
    print(x(i))          <=  this prints 3 times
END
```

Output:

```
1          i=1, first element of x
2          i=2, second element of x
3          i=3, third element of x
```

FOR Loop

```
x=[1 2 3]
a=length(x)
FOR i=1:a
   print(x(i))
END
```



FOR Loops

- FOR: if you know how many times it should execute

```
x=[1 2 3]
a=length(x)              <= a now is 3, not null
FOR i=1:a                <= i is 1, then 2, then 3.
    print(i)             <=  this prints 3 times
    print(x(i)*x(i))     <=  this prints 3 times
END
```

Output:

1	Value of i first time through
1	Value of x(i)*x(i) when i=1
2	Value of i second time
4	Value of x(i)*x(i) when i=2
3	Value of i third and final time
9	Value of x(i)*x(i) when i=3

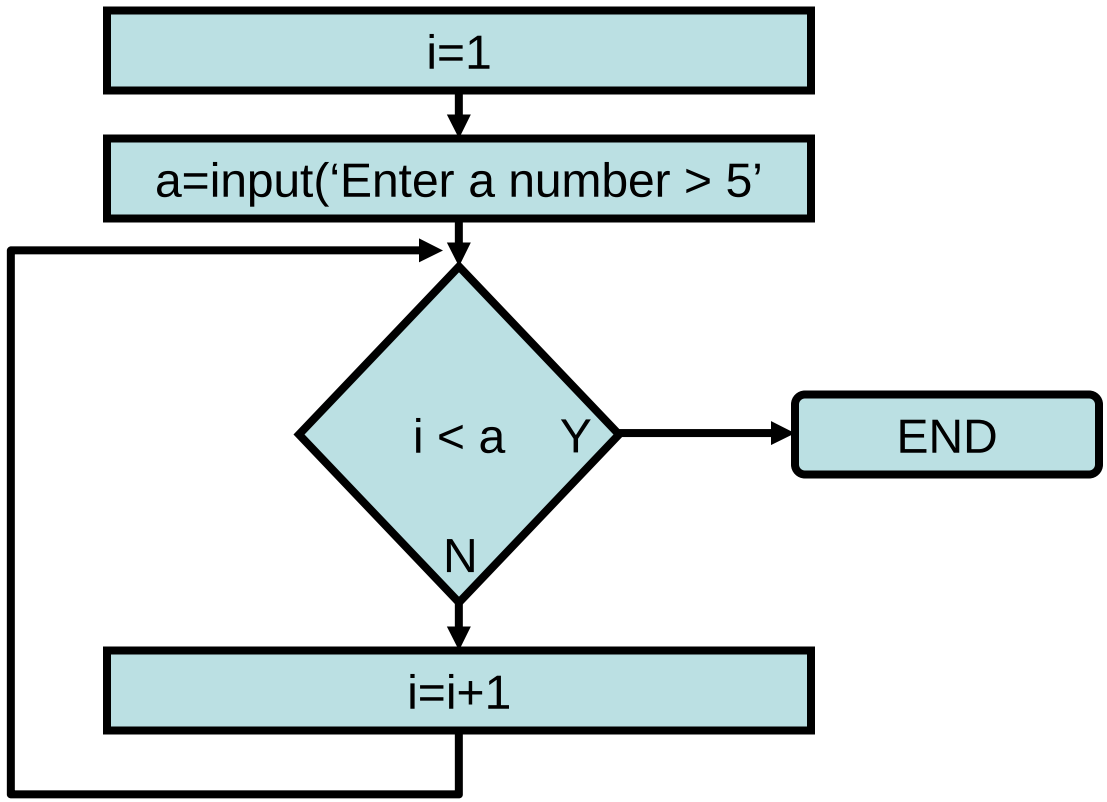
WHILE Loops

- When you don’t know how many times it must loop
- Must loop until converges, or until told to quit
- Can loop forever if not careful!!

```
i=1
a=input('Enter a number > 5'
WHILE ( i < a)
    i=i+1
END
```

While Loop

i=1
a=input('Enter a number > 5'
WHILE (i < a)
    i=i+1
END



Multiple things in a FOR, WHILE, or IF

- Usually, you can do multiple commands as part of a FOR, WHILE, or IF.

```
x=[1 2 3]
a=length(x)
FOR i=1:a
    print(i)
    print(x(i)*x(i))
    print('Done with this loop iteration')
END
```

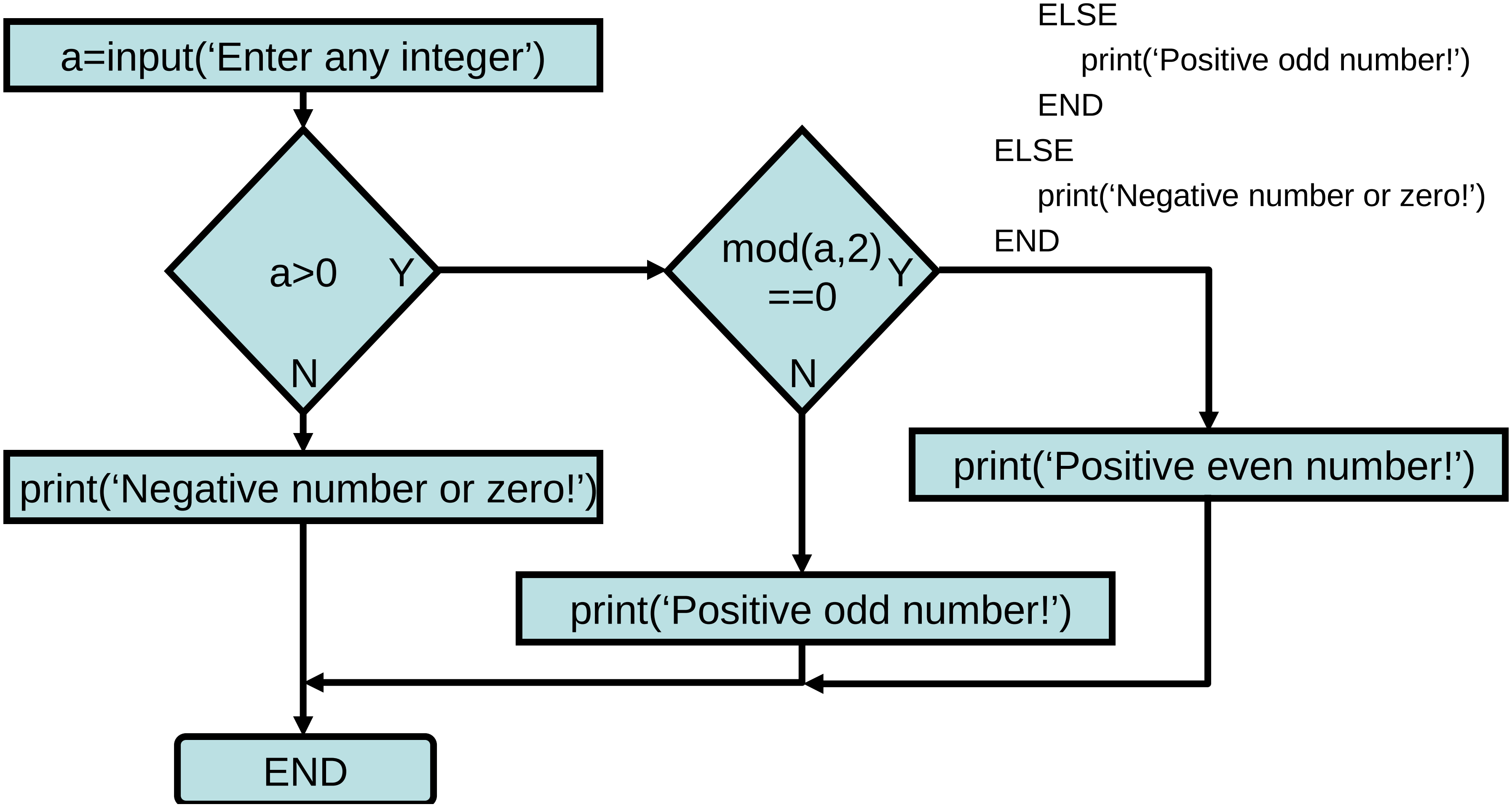
- IF inside IF? FOR inside FOR?

Nested IF

```
a=input('Enter any integer')
IF ( a > 0 )
    IF ( mod(a,2) == 0 )
        print('Positive even number!')
    ELSE
        print('Positive odd number!')
    END                    (This ends the inner IF statement)
ELSE
    print('Negative number or zero!')
END                        (This ends the outer If statement)
```

Nested IF

```
a=input('Enter any integer')
IF ( a > 0 )
    IF ( mod(a,2) == 0 )
        print('Positive even number!')
    ELSE
        print('Positive odd number!')
    END
ELSE
    print('Negative number or zero!')
END
```



Nested FOR

```
FOR row=1:3
    FOR col=1:3
        x(row,col) = row*col
    END                                   (Ends inner FOR statement)
END                                       (Ends inner FOR statement)
```

row	col	*(Col is inner loop, iterates through repeatedly)*
1	1	
1	2	
1	3	
2	1	
2	2	
2	3	
3	1	
3	2	
3	3	

Resulting x is a 2 dimensional array:

$$x = \begin{array}{ccc} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{array}$$

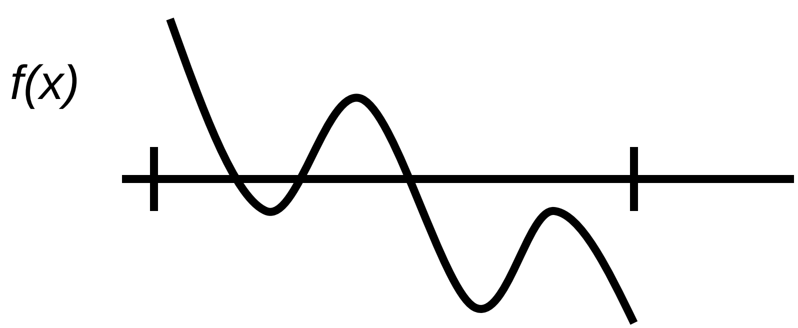
Subroutines

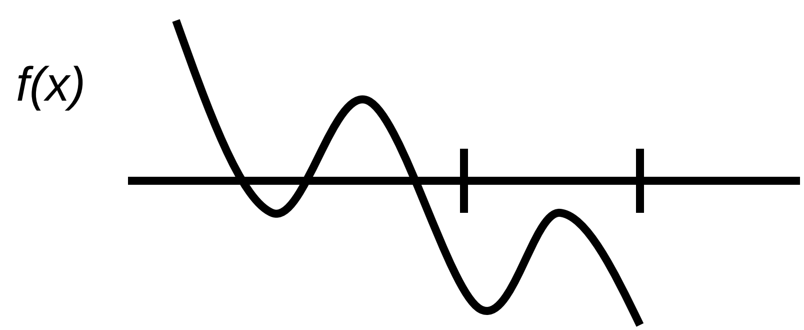
- Sometimes you need to do a few things over and over
- Do not replicate same code over and over
- Make a user defined function

```
FUNCTION  out=average(input)
sum=0                          (Initialization is important!)
a = length(input)
FOR i=1:a
    sum=sum+input(i)           (Counter or summer is important)
END
out = sum / a
```

Bisection Method

- Given a function of one variable *x*, find a value of *x* such that $f(x)=0$.
- Assume that you have upper and lower bounds on *x*
- Assume the function is continuous on your domain
- Assume that $f(x_l)$ and $f(xu)$ are different signs (+/- or -/+)
  - There is at least one zero crossing on your domain





Bisection Method Pseudo Code

```
Get xl and xu
Determine xm, the midpint using bisection
Determine f(xl), f(xu), f(xm)
WHILE f(xm)^2 > epsilon     (while not converged)
    IF f(xl) and f(xu) both positive or both negative
        xl=xm
    ELSE
        xu=xm
    END
    Determine new xm and f(xm)
END
```

Bisection Method Pseudo Code With error checking

```
Get xl and xu
Determine xm, the midpint using bisection
Determine f(xl), f(xu), f(xm)
IF f(xl) and f(xu) both positive or negative OR if (xl>=xu)
      Display error message and quit
END
WHILE f(xm)^2 > epsilon     (while not converged)
      IF f(xl) and f(xu) both positive or both negative
            xl=xm
      ELSE
            xu=xm
      END
      Determine new xm and f(xm)
END
```

```
% Initialization of epsilon, xl, xu
epsilon = 0.00001
xl=input('Enter lower bound on x')
xu=input('Enter upper bound on x')
IF ( xl >= xu)
    print('Error, xl >= xu)
    return
END
xm= xl + (xu-xl)/2
fxl=nonlinearfunction(xl)
fxu=nonlinearfunction(xl)
fxl=nonlinearfunction(xl)
IF (fxl*fxu > 0)
      print('f(xl) and f(xu) are both positive or negative')
      return
END
```

```
WHILE ( fxm*fxm  > epsilon )
       IF  (  fxl*fxm > 0 )
                xl = xm
                fxl = fxm
        ELSE
                xu = xm
                fxu = fxm
        END
        xm = xl + (xu-xl) / 2
        fxm=nonlinearfunction(xm)
END
print (‘solution for f(x)=0 is:’)
print (xm)
```

Debugging

- You will make errors
- Try to go through procedure step by step
- Visualize what variables have what values at each step
- **Add print statements to print out values**
- Get parts working as you expect, then build on that
- Try smaller cases, then extrapolate
  - FOR i=1:3 instead of FOR i=1:length(X)
- Use good variable names (depends on language)
  - Balance between long and short, clarity and typing
  - i instead of first_loop_integer_counter
  ListOfStudentGrades or list_of_grades or list
- Add comments to your code!
- Try "edge cases" to see if anything breaks

My Method

- Get some code to do something (small part of whole)
- Test code to see if it runs as expected
- Expand code to do something else
- Repeat

- Don’t just sit down and write a program and hope it works the first time. Do sections, parts, so you always have something that works.
- Keep versions. Periodically make a backup copy: program-3.m
- After you get it to work, you may need to step back and redo the whole thing (once you finish, a simple easy way may be apparet)