Programming, control structures

Loriano Storchi

loriano@storchi.org

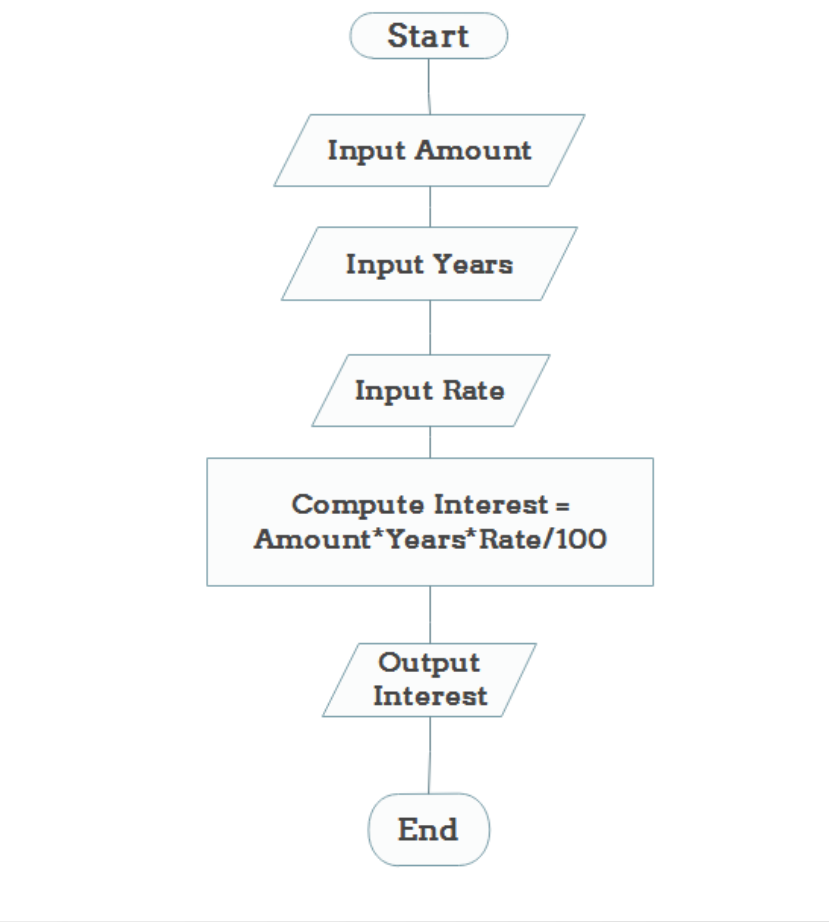
http::://www.storchi.org/

FLOWCHART AND PSEUDOCODE

FLOWCHART AND PSEUDOCODE

- We will only go through some basic example: Calculate the Interest of a Bank Deposit

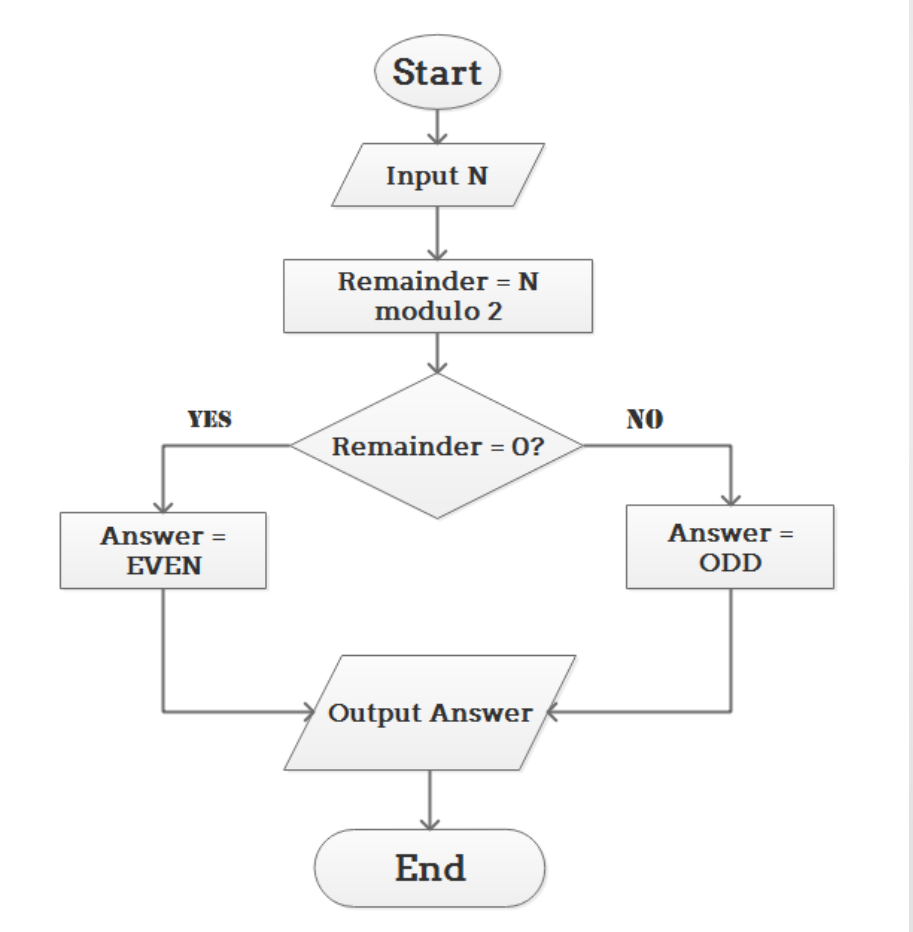
Step 1: Read amount,
Step 2: Read years,
Step 3: Read rate,
Step 4: Calculate the interest with formula
"Interest=Amount*Years*Rate/100
Step 5: Print interest,



FLOWCHART AND PSEUDOCODE

- Determine and Output Whether Number N is Even or Odd

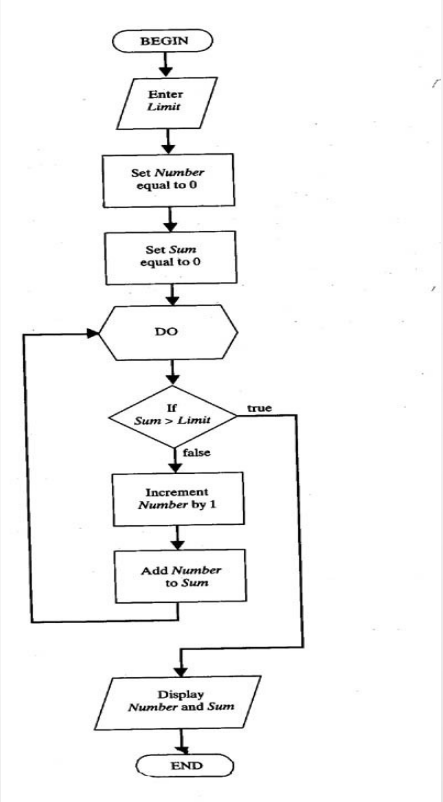
Step 1: Read number N,
Step 2: Set remainder as N modulo 2,
Step 3: If remainder is equal to 0 then number N is even, else number N is odd,
Step 4: Print output.



FLOWCHART AND PSEUDOCODE

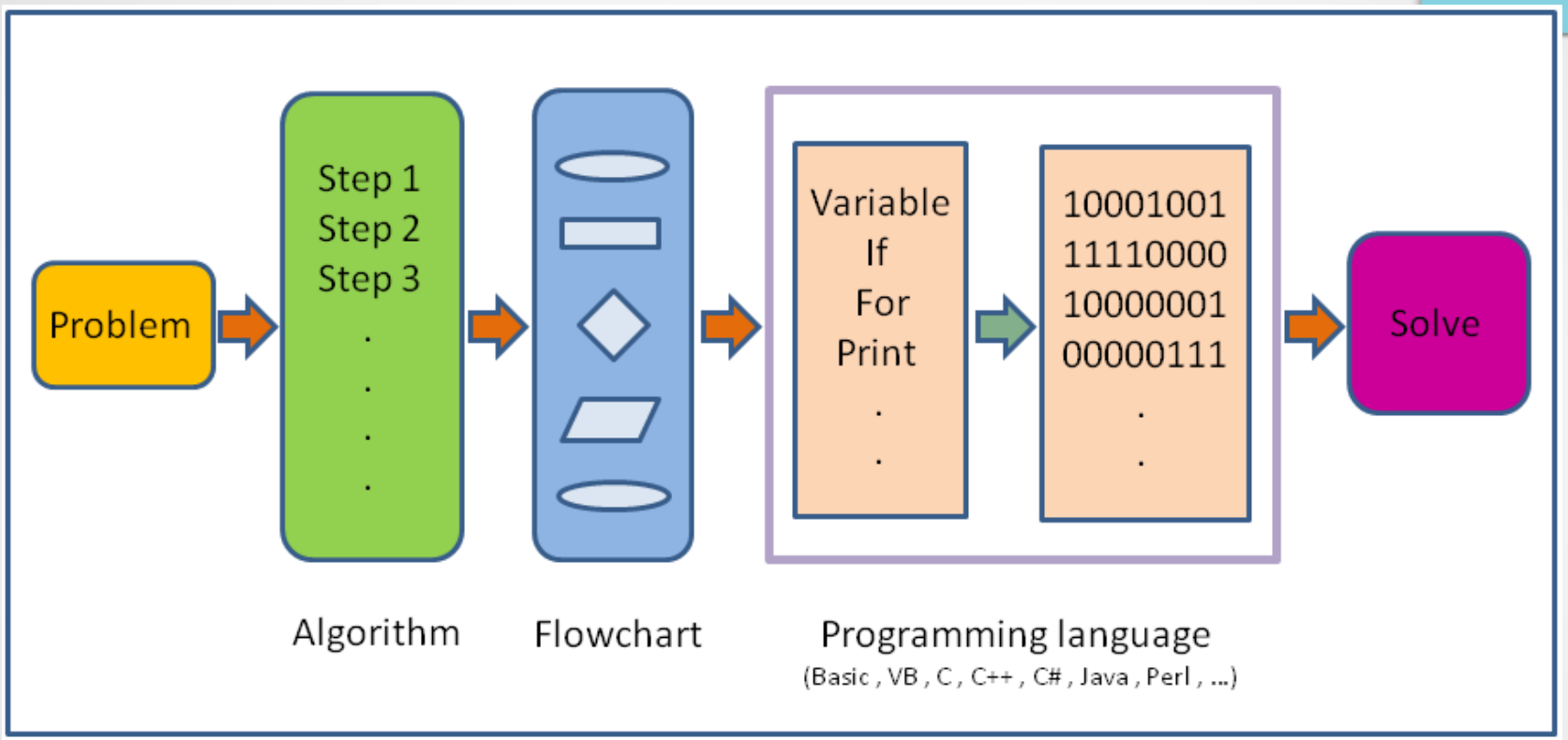
- For a given value, Limit, what is the smallest positive integer **Number** for which the sum Sum = 1 + 2 + ... + Number **is greater than Limit. What is the value for this Sum?**

1. Enter Limit
2. Set Number = 0.
3. Set Sum = 0.
4. Repeat the following:
   a. If Sum > Limit, terminate the repitition, otherwise.
   b. Increment Number by one.
   c. Add Number to Sum and set equal to Sum.
5. Print Number and Sum.



CONTROL STRUCTURES

CONTROL STRUCTURES



There are three fundamental structures that are used for the algorithmic resolution of problems: **selection, iterations and sequence** (sequence of instructions) (the GOTO present in machine languages since the 70s has been progressively discouraged / eliminated)

Examples: https://bitbucket.org/lstorchi/teaching
https://github.com/lstorchi/teaching

SEQUENCES

Sequences

- **Sequence control structure** refers to the line-by-line execution by which statements are executed sequentially, in the same order in which they appear in the program. They might, for example, carry out a series of read or write operations, arithmetic operations, or assignments to variables.

Step 1: Read amount,
Step 2: Read years,
Step 3: Read rate,
Step 4: Calculate the interest with formula
"Interest=Amount*Years*Rate/100
Step 5: Print interest,

SELECTION

Selections

- The general structure of a selection instruction is as follows:

if (condition 1)

statements 1

else if (condition 2)

statements 2

...

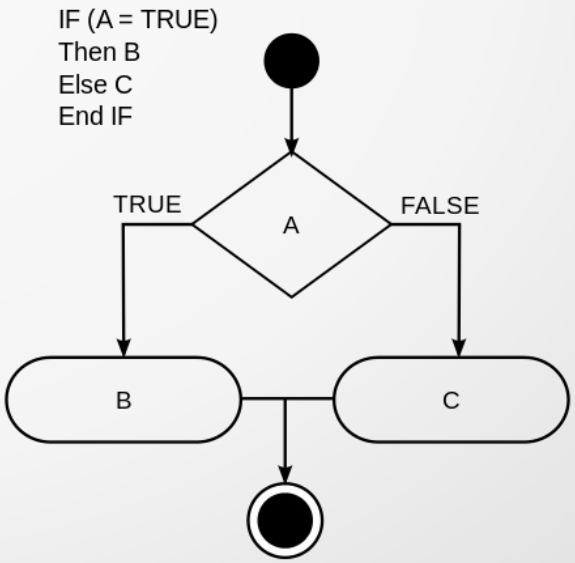
else

statements N

endif

There can be so many else if

They do not necessarily have to show all three elements, if, else if and else, I can also have just a single if



Selections

- **Nesting**, I can nest the selection instructions of course:

```
if (condition 1)
  statement 1
  if (condituion 2)
    statements 2
  end if
else
  statements 3
end if
```

Operators

- In all programming languages, I can use relationship operators to compare numbers and variables, for example:

Description	Java, C, C++	Fortran
Greater than	>	.GT.
Greater than or equal	>=	.GE.
Less than	<	.LT.
Less than or equal	<=	.LE.
Equal	==	.EQ.
Not Equal	!=	.NE.

```
Int a;

a = 4 // operatori di  assegnazione

If  (a == 5)
{
   cout << “Hello” << std::endl;
}
else if (a > 5)
{
    cout <<  a << “ > 5 “ << std::endl;
}
```

Logical Operators

- I take for granted the logical operators AND, OR and NOT

Logical Operators		
Operator	Description	Example
&&	AND	x=6 y=3 x<10 && y>1 Return True
||	OR	x=6 y=3 x==5 || y==5 Return False
!	NOT	x=6 y=3 !(x==y) Return True

Examples the following expression

5 < a < 7

In programming languages it is broken into two elementary expressions linked by the operator AND:

(a > 5) AND (a < 7)

Bitwise operations

- Do not confuse the previous with the bitwise operations
- These are the operations that are used to manipulate bitwise data, not to be confused with qunto seen in the previous slide
  - & (AND)
  - | (OR)
  - ^ (XOR I.e. 1 XOR 1 is zero)
  - ~ (ones' complement i.e. 0 to 1 and 1 to 0)
  - >> (right shift 11100101 >> 1 is 01110010)
  - << (left shift)

Bitwise operations

- A simple test just to clarify

```
└─ $ python3
Python 3.6.9 (default, A
[GCC 8.4.0] on linux
Type "help", "copyright"
>>> a = 60
>>> b = 13
>>> print(a&b)
12
>>> exit()
```

```
a = 60
b = 13

print(a&b)
```

```
12
```

```
a = 0011 1100

b = 0000 1101

---------------

a&b = 0000 1100
```

Case structure

- Basically a series of if-then-else with some constraint. In practice, the choice between the blocks of instruction is guided by the value of a certain variable:

```
switch variable:
   case val1:
      statements 1
   case val2:
      statements 2
   …
   default:
     default statements
```

```
If (variable == val1)
{
    Statements 1
}
else if (variables == val2)
{
   Statements 2
}
…
else
{
  Default statements
}
```

Example

```
int i;
i = 4;
switch (i)
{
  case 1:
    printf("vale uno \n");
    break;
  case 2:
    printf("vale due \n");
    break;
  default:
    printf("non uno non due\n");
    break;
}
```

LOOPS

Loops

- There are three different types: **while..do** , **do..while** e **for**
- **Not all languages necessarily have all three of these structures**
- **These structures allow to repeat a block of instructions until a condition occurs**
- Also in this case it is possible to **nest** more loops one inside the other

While..do e Do..while

- The block of instructions can also never be executed, since **the condition is checked at the beginning**, as long as the condition is true, the block of instructions is executed

  WHILE (condition)

  statements

  END DO

- do..while instead **executes the block of instructions until the condition is false**

  DO

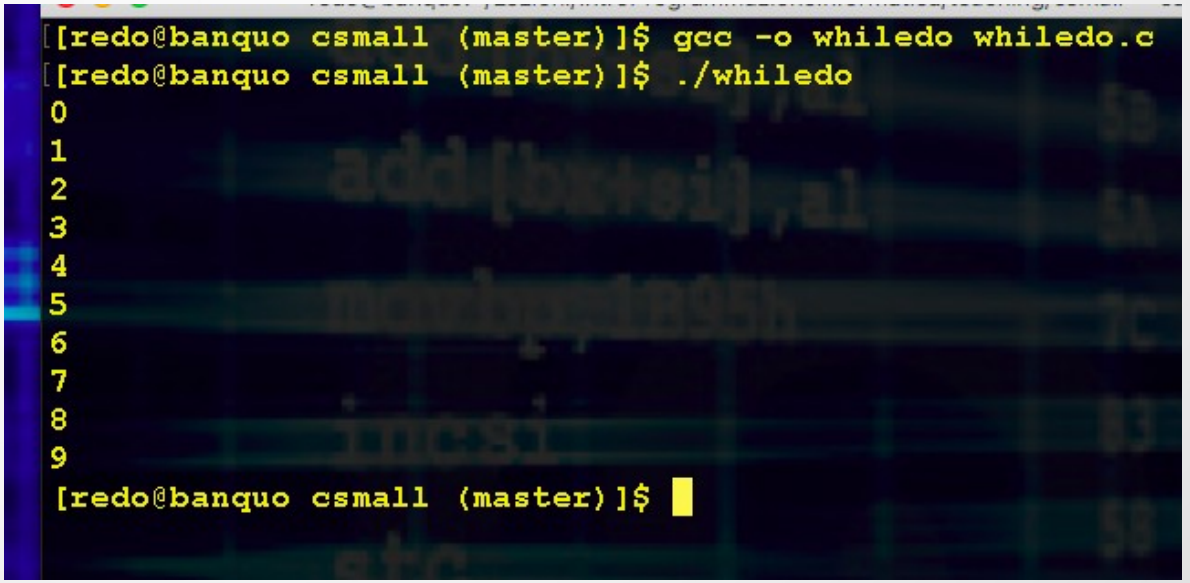
  statements

  WHILE (condition)

Example

- A simple C example:

```
int i = 0, N = 10;
while (i != N)
{
    printf("%d \n", i);
    i++;
}
```



Example

```
Type "help", "copyright"
>>> while True:
...     print("here")
...
```

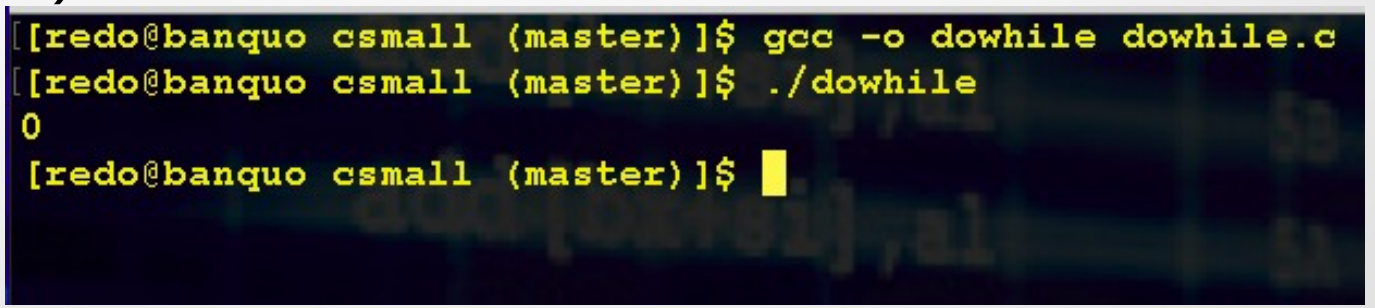
```
while True:
    print("here")
```

```
here
here
here
here
here
here
here
here
here
here
here
```

Example

```
Int i = 0, N = 10;


do
{
    printf (“%d \n”, i);
    i++; // i = i + 1
} while (i >= N);
```



For loop

- It executes a block of instructions a number of times known from the beginning. Many programming languages force the programmer to use a counter.
- Generally there is a counter which is an integer variable whose value is "changed" step by step
- The end condition is generally determined by comparing the counter variable with a value

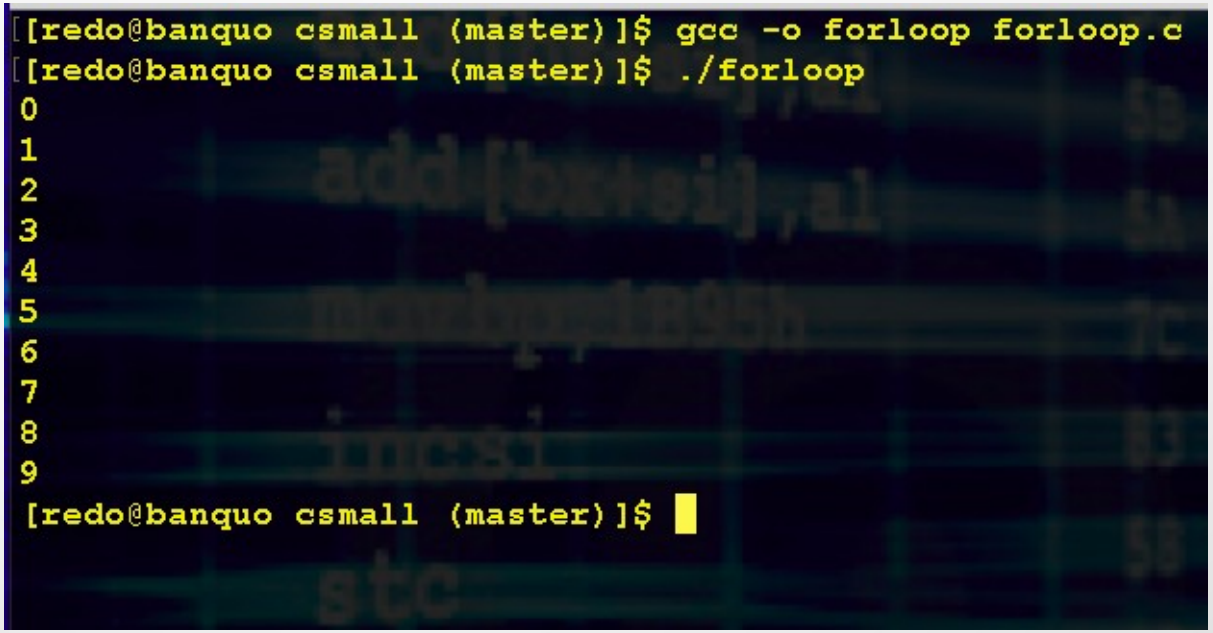
for (counter = startingvalue A endvalue STEP = stepvalue)

statements

end for

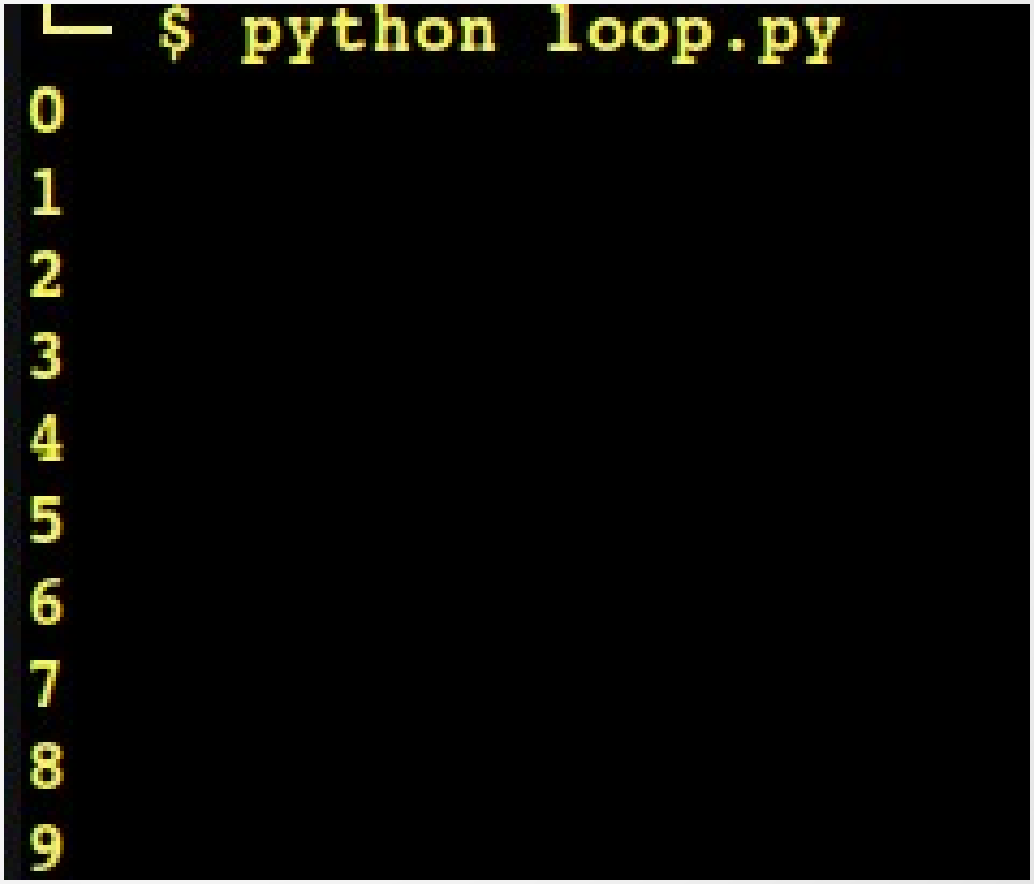
Example

```
int i;
for (i=0; i<10; ++i)
{
   printf (“%d \n”, i);
}
```



Example

```
for i in range(10):
    print(i)
```

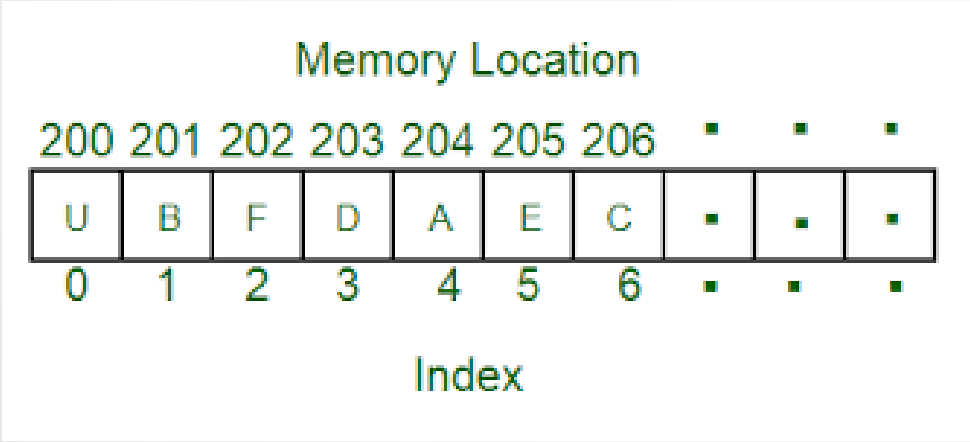


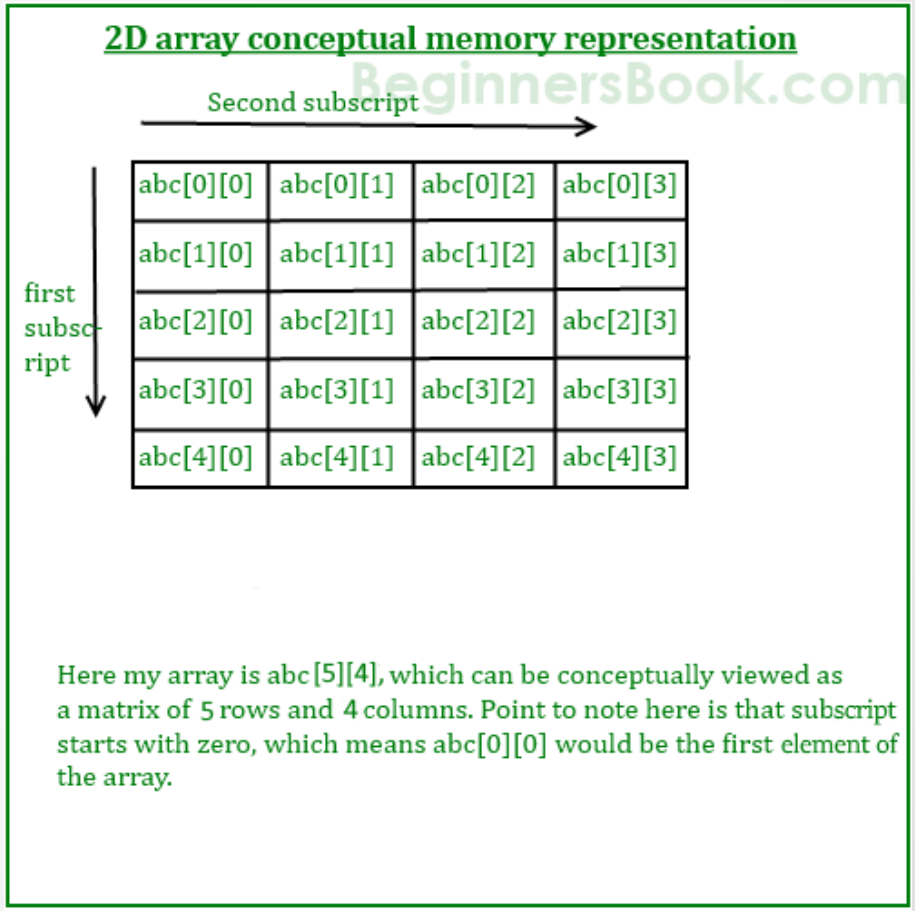
ARRAY

Array

- How can I easily handle information that is structured by nature? For example a complex number, or a vector or a matrix?
- Typical structured variables are arrays
  - For example, a vector of floating-point numbers in C can be declared as: float v [10];
  - We can then access the i-th element of the vector: v [i-1] = 3.5;
- Likewise I can define an array (two-dimensional array) as: float m [10] [10];

Array

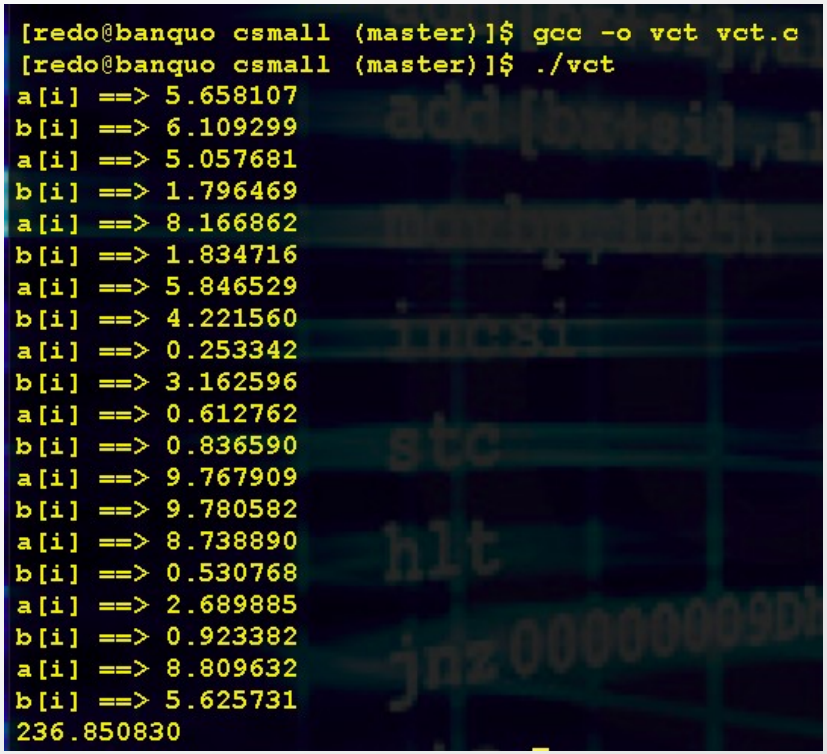

Memory Location
200 201 202 203 204 205 206
U B F D A E C
0 1 2 3 4 5 6
Index


2D array conceptual memory representation
BeginnersBook.com
Second subscript
first subscript
abc[0][0] abc[0][1] abc[0][2] abc[0][3]
abc[1][0] abc[1][1] abc[1][2] abc[1][3]
abc[2][0] abc[2][1] abc[2][2] abc[2][3]
abc[3][0] abc[3][1] abc[3][2] abc[3][3]
abc[4][0] abc[4][1] abc[4][2] abc[4][3]
Here my array is abc[5][4], which can be conceptually viewed as a matrix of 5 rows and 4 columns. Point to note here is that subscript starts with zero, which means abc[0][0] would be the first element of the array.

Example

- Example loops to perform a scalar multiplication between vectors

```
float a[N], b[N];
for (i=0; i<N; ++i)
{
  a[i] = (float)rand()/(float)(RAND_MAX/N);
  b[i] = (float)rand()/(float)(RAND_MAX/N);
}
s = 0.0;
for (i=0; i<N; ++i)
{
    s = s + a[i]*b[i];
}
```



Example

```
import random

N = 20

a = []
b = []

for i in range(N):
    a.append(random.randrange(1, 30))
    b.append(random.randrange(1, 30))

for i in range(N):
    print("%3d %3d"%(a[i], b[i]))

s = 0.0
for i in range(10):
    s = s + float(a[i]*b[i])

print("")
print("S = %.5e"%s)
```