This presentation is available at:
www.lu.fme.vutbr.cz/~schor

Institute of Aerospace engineering,
Faculty of mechanical engineering,
Brno University of technology

Pavel Schor

Brno
June 2012


```c
#include <stdio.h>
#include <malloc.h>

int main(void)
{
    int i, n;
    n = 3;
    int a[2];
    double *b;
    b = (double*) malloc(n*sizeof(double));
    printf("\n\n%s\n","= Example 1 =");
    printf("\n\n%s \n","Size of integer is", sizeof(int));
    printf("\n\n%s \n","Size of double is", sizeof(double));

    a[0]=1;
    a[1]=2;
    printf("\n\n%s\n","// Integer array a[2]:");
    printf("\n\n%s %d %s %u \n","a[1]: Value is ", a[0]," Adress is ", &a[0]);
    printf("\n\n%s %d %s %u \n","a[2]: Value is ", a[1]," Adress is ", &a[1]);

    for (i=0; i<n; i++)
    {
        b[i]=(double)i;
    }
    printf("\n\n%s%d%s\n","// Double array b[\n,n\n]:");
    for (i=0; i<n; i++)
    {
        printf("\n\n%s%d%s %f %s %u\n","b[\n,i\n]: Value is ",*(b+i)," Adress is ",b+i);
    }
} 
```
Short introduction for non-programmers

Variable:
- address is managed by the operating system / program environment

Reference: C PROGRAMMING LANGUAGE, Kerninghan, Ritchie

```
pavel@pavel-laptop: ~/Dokumenty/OLU/catPy

-------------- Example 1 ---------------
Size of integer is: 4
Size of double is: 8

///// Integer array a[2]:
a[1]: Value is 1 Address is 3218059164
a[2]: Value is 2 Address is 3218059168

///// Double array b[3]:
b[0]: Value is 0.000000 Address is 144662536
b[1]: Value is 1.000000 Address is 144662544
b[2]: Value is 2.000000 Address is 144662552
pavel@pavel-laptop:~/Dokumenty/OLU/catPy$ ./a.out

-------------- Example 1 ---------------
Size of integer is: 4
Size of double is: 8

///// Integer array a[2]:
a[1]: Value is 1 Address is 3213076732
a[2]: Value is 2 Address is 3213076736

///// Double array b[3]:
b[0]: Value is 0.000000 Address is 147939336
b[1]: Value is 1.000000 Address is 147939344
b[2]: Value is 2.000000 Address is 147939352
pavel@pavel-laptop:~/Dokumenty/OLU/catPy$ 
```
Short introduction for non-programmers

Structures:
- A lot of variables makes program bad readable for humans
- This usually leads to errors, prolongs program development, ...

Solution is called data structures:
- Data with common denominator are grouped together

- dot convention is used:

  Parent.child=something

Example in GNU/octave, MATLAB:

Aircraft=struct()
Wing=struct()
Wing.parameters=struct()

Wing.parameters.span=10.0
Wing.parameters.rootChord=0.9
Wing.parameters.tipChord=0.35
Wing.parameters.airfoil = „NACA 2412“

Aircraft.wing=Wing
Short introduction for non-programmers

Functions:
- A lot of lines makes program bad readable for humans
- This usually leads to errors, prolongs program development, ...

Solution is called functions:
- Repeated calculations are done by functions

```
returnValue=function(input1,...inputN)

Example in GNU/octave, MATLAB:

function m=getVectorMagnitude(v)
    m=( v(1)^2 + v(2)^2 + v(3)^2 )^0.5;
end

a=[1,0,0]
vm=getVectorMagnitude(a)
```
Short introduction for non-programmers

Objects:
- Contains data and functions
- Functions are called “methods“
- Object is an instance of a class
- Class is a template, which defines the methods and how the data are stored
- Dot convention is usually used:

```python
object.data = something

something = Object.method(input1, .. inputN)
```
Short introduction for non-programmers

Python:

- free and open source software
- a general-purpose, high-level programming language
- multi-paradigm programming language, intended to be a highly readable
- large standard library, providing pre-written tools suited to many tasks. NumPy + SciPy can easily replace Matlab, except toolboxes.
- mainly used as a scripting language, but Python code can be packaged into standalone executable programs.
- interpreters are available for many operating systems.
- uses whitespace indentation, rather than braces or keywords, to delimit blocks.
Short introduction for non-programmers

Objects:

- An example in python:

```python
class myNumber:
   def __init__(self):
       self.value=0.0  # constructor method  # data

   def setValue(self,value):
       self.value=value  # some other methods

   def getValue(self):
       return self.value

   def square(self):
       return self.value*self.value

   def multiply(self,n):
       return self.value*n

number1=myNumber()  # number1 is an instance of the myNumber class
number1.setValue(5.0)  # number1.value = 5.0
b=number1.multiply(10.0)  # b = 5*10
```
Short introduction for non-programmers

Objects:
- An example in python:

```python
pavel@pavel-laptop:~/Dokumenty/OLU/catPy$ ipython myObjects.py
Python 2.6.6 (r266:84292, Dec 27 2010, 00:02:40)
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?        -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help     -> Python's own help system.
object?  -> Details about 'object'. ?object also works, ?? prints more.

In [1]: b
Out[1]: 50.0

In [2]: number1
Out[2]: <__main__.myNumber instance at 0xb6fecf0c>

In [3]: number1.getValue()
Out[3]: 5.0

In [4]: 
```
Scripting the CATIA V5

Requirements:
- MS Windows operating system
- Access to WindowsAPI
- CATIA V5, V6
- Text editor, not Word, Notepad

WindowsAPI:
- API = Application Programming Interface
- Collection of objects, which can be used by programmer

- Examples:
  
  Program can run another program by calling WinAPI

  Windows dialog boxes are usually created by calling the WinAPI

  Programs can interact each other using the WinAPI + COM
COM:

- Component Object Model
- Microsoft technology for inter-process communication
- Programs can create new objects or modify existing objects in other programs.

Example:

Spell check is used in word processor and in e-mail application.

Instead of creating two spell checks, only spell check in the word-processor is created.

If there is a need for e-mail spell checking, the spell check object from the word processor is called via WinAPI.

The binary interface, which allows this ability is called COM

- Interacting programs can be written in different languages.
- Common thing is the same object in all programs.
Strategy for working with CATIA from script:

- The strategy for working with CATIA from the script is same as the strategy for working with CATIA using the GUI.

- COM objects require same inputs as the user fills in GUI forms.

- However the task is more complex, for example naming objects in CATIA bodies may be necessary for using references.

What to do? Getting help:

- CATIA has build-in VB engine, which can record macros:
  
  menu \rightarrow Tools \rightarrow Macro \rightarrow Start recording

- Reading recorded macros is fast method for understanding, what is all that about, but recorded macros can be filled with ballast stuff.

- Help is available:
  Dassault Systemes/B19/intel_a/code/bin/V5Automation.chm
Description of CATIA objects:
Scripting the CATIA V5 Infrastructure Automation Objects:
Scripting the CATIA V5

Part Document Objects:

PartDocument
  Part
    OriginElements
      PlaneXY, Plane
      PlaneYZ, Plane
      PlaneZX, Plane
    AxisSystems
      AxisSystem
    OrderedGeometricalSets
      OrderedGeometricalSet
    GeometricElements
      GeometricElement
    Bodies
      Body
        Sketches
          Sketch
        Shapes
          Shape
          Boundary
        HybridBodies
    HybridBodies
      HybridBody
      HybridShapes
        HybridShape
        Boundary
      Sketches
        Sketch
      GeometricElements
        GeometricElement
      Bodies
        HybridBodies
    Constraints
      Constraint
      Relations
        Relation
      Parameters
        Parameter
      Factory
        ShapeFactory
        HybridShapeFactory
        InstanceFactory
      AnnotationSets
      UserSurfaces

Legend
- Collection
- Abstract object
- Object
Creating new part

```python
# Binding python session into CATIA
import win32com.client.dynamic  # Module for COM-Client
CATIA = win32com.client.Dispatch("CATIA.Application")

# CATIA object for managing documents
documents1 = CATIA.Documents

# Starting new part
partDocument1 = documents1.Add("Part")
part1 = partDocument1.Part
```

Creating new body

```python
# Starting new body (geometrical set) in part1
bodies1 = part1.HybridBodies

# Adding new body to part1
body1 = bodies1.Add()

# Naming new body as "wireframe"
body1.Name="Wireframe"
```
Starting new shape factory

Interface to create all kinds of HybridShape objects that may be needed in wireframe and surface design.

```
# Shape factory provides generating of shapes
ShFactory = part1.HybridShapeFactory

# Creating new point [0,0,0] in Wireframe
point0 = ShFactory.AddNewPointCoord(0.000000, 0.000000, 0.000000)
body1.AppendHybridShape(point0)

# part1 should be updated after every new object
part1.Update()
```
Example 1: System model of tapered wing

Actions are the same as in interactive session!
Example1: System model of tapered wing

```python
import win32com.client.dynamic  # Module for COM-Client
import sys, os               # Module for File-Handling
import win32gui              # Module for MessageBox
import numpy as np            # Module for numerical computing

# Some basic geometrical data
halfSpan=1000.0
rootLength=100.0
tipLength=50.0
rootTwist=0.0
tipTwist=5.0

# Binding python session into CATIA
CATIA = win32com.client.Dispatch("CATIA.Application")
documents1 = CATIA.Documents                # CATIA object for managing documents
partDocument1 = documents1.Add("Part")     # Starting new part
part1 = partDocument1.Part

#Shape factory provides generating of shapes
ShFactory = part1.HybridShapeFactory       # Starting new body (geometrical set) in part1
bodies1 = part1.HybridBodies               # Adding new body to part1
body1 = bodies1.Add()                      # Naming new body as "wireframe"
body1.Name="Wireframe"
```
Example1: System model of tapered wing

```python
bodies2 = body1.hybridBodies  # Starting new geometrical set in Wireframe
body2 = bodies2.Add()         # Adding new body to Wireframe
body2.Name="RootSection"     # Naming new body as "RootSection"
body3 = bodies2.Add()         # Adding new body to Wireframe
body3.Name="TipSection"

body4 = bodies1.Add()         # Adding new body in part1
body4.Name="Surfaces"        # Naming new body as "Surfaces"

# Loading point coordinated from text file
RootAirfoil=np.loadtxt('data/clarky.dat',skiprows=1)
TipAirfoil=np.loadtxt('data/clarky.dat',skiprows=1)

# Creating new point \([0,0,0]\) in Wireframe
point0 = ShFactory.AddNewPointCoord(0.000000, 0.000000, 0.000000)
body1.AppendHybridShape(point0)  # part1 should be updated after every new object
part1.Update()

# Creating Z-direction for translating wing sections
wingAxis1= ShFactory.AddNewDirectionByCoord(0.000000, 0.000000, 1.000000)

# Creating twist point, sections will be twisted around this point
twistPoint1=ShFactory.AddNewPointCoord(25.0,0.0,0.0)
twistRef1= part1.CreateReferenceFromObject(twistPoint1)
```
Example 1: System model of tapered wing

# Creating Z-direction for translating wing sections
twistDir1 = ShFactory.AddNewDirectionByCoord(0.000000, 0.000000, 1.000000)

# Creating [POINT-DIRECTION] axis for twisting wing sections
twistAxis1 = ShFactory.AddNewLinePtDir(twistRef1, twistDir1, 0.000000, 20.000000, False)

# Starting new spline for root section
spline1 = ShFactory.AddNewSpline()
spline1.SetSplineType(0)
spline1.SetClosing(0)

# Filling the spline with points
for i in range(0, len(RootAirfoil)):
    PT = RootAirfoil[i] * 100  # coordinates are 0..1 which is too small for CATIA
    point = ShFactory.AddNewPointCoord(PT[0], PT[1], 0.0)  # coordinates are 2D, Z=0.0
    spline1.AddPoint(point)  # new point to spline is added
ShFactory.GSMVisibility(spline1, 0)  # hide the spline
Example1: System model of tapered wing

```python
# Starting new spline for tip section
spline2 = ShFactory.AddNewSpline()
spline2.SetSplineType(0)
spline2.SetClosing(0)

# Filling the spline with points
for i in range(0, len(TipAirfoil)):
    PT = TipAirfoil[i] * 100
    point = ShFactory.AddNewPointCoord(PT[0], PT[1], 0.0)
    spline2.AddPoint(point)
ShFactory.GSMVisibility(spline2, 0)
```
Example 1: System model of tapered wing

```plaintext
#Scale [REFERENCE POINT - RATIO] the root section
ref1 = part1.CreateReferenceFromObject(spline1)
ref2 = part1.CreateReferenceFromObject(twistPoint1)
scaling1 = ShFactory.AddNewHybridScaling(ref1,ref2, rootLength/100.0)
scaling1.VolumeResult = False
body2.AppendHybridShape(scaling1)
ShFactory.GSMVisibility(scaling1,0)

#Rotate [AXIS] the root section
rotate1= ShFactory.AddNewEmptyRotate()
ref1= part1.CreateReferenceFromObject(scaling1)
ref2 = part1.CreateReferenceFromObject(twistAxis1)
rotate1.ElemToRotate = ref1
rotate1.VolumeResult = False
rotate1.RotationType = 0
rotate1.Axis = twistAxis1
rotate1.AngleValue = rootTwist
body2.AppendHybridShape(rotate1)
ShFactory.GSMVisibility(rotate1,0)
```
Example1: System model of tapered wing

#Translate [DIRECTION - DISTANCE] the root section
# is actually not necessary here
translate1 = ShFactory.AddNewEmptyTranslate()
ref1 = part1.CreateReferenceFromObject(rotate1)
translate1.ElemToTranslate = rotate1
translate1.VectorType = 0
translate1.Direction = wingAxis1
translate1.DistanceValue = 0.00
translate1.VolumeResult = False
translate1.Name = "rootShape"  # Naming result "rootShape" IMPORTANT!!!
body2.AppendHybridShape(translate1)

Create the tip section yourself
Example 1: System model of tapered wing

```python
# Create new loft - MULTISECTION SURFACE
loft1 = ShFactory.AddNewLoft()
loft1.SectionCoupling = 1
loft1.Relimitation = 1
loft1.CanonicalDetection = 2

# Adding root section to the loft
shapes1 = body2.HybridShapes
# getting item from pool!!
result1 = shapes1.Item("rootShape")
ref1 = part1.CreateReferenceFromObject(result1)
ref2 = None
loft1.AddSectionToLoft(ref1, 1, ref2)

# Adding tip section to the loft
shapes2 = body3.HybridShapes
# getting item from pool!!
result2 = shapes2.Item("tipShape")
ref1 = part1.CreateReferenceFromObject(result2)
ref2 = None
loft1.AddSectionToLoft(ref1, 1, ref2)
loft1.Name = "masterSurface"

# Adding loft to Surfaces geometrical set
body4.AppendHybridShape(loft1)
part1.Update()
```