Applied Programming and Computer Science, DD2325/appcs14

PODF, Programmering och datalogi för fysiker, DA7011

Autumn 2014

A. Maki, C. Edlund
atsuto@kth.se

Course Information
https://www.kth.se/social/course/DD2325/

Goal

– To improve the programming technique, and
– To gain basic knowledge about program and data structures.

Who are teaching?

- **Atsuto Maki**, CSC/KTH
- **Niyazi Cem Degirmenci**, Teaching Assistant, CSC/KTH
- **Alex Loiko**, Teaching Assistant, CSC/KTH
- **Fredrika Agestam**, Teaching Assistant, CSC/KTH
- **Carina Edlund**, Administration Assistant, CSC/KTH

The course contents are given through:

- Lectures
- Exercises/Labs (Primary contact: ncde@kth.se)

NB! Do register to the course, and to the exam.

After completing the course the student should be able to

- write structured programs in Matlab and small programs C
- do systematic error search in programs
- describe and use different data types
- use abstraction as a tool to simplify programming
- compare algorithms with respect to time and memory needs, complexity
- describe algorithms for searching and sorting
- formulate and implement recursive algorithms
- implement and use stacks, queues, trees, hash tables and hash functions
- describe fundamental algorithms for compression

Examination

The examination in this course consists of two parts:

1. written exam in January (TEN1; 3 cr)
   Grade A,B,C,D,E,FX,F.
2. computer assignments (LAB1; 4.5 cr).
   Mandatory. Grade P/F.

Computer assignments include:

1. Evaluation Using Reverse Polish Notation
2. Debugging in MATLAB and A Quicksort Implementation
3. Newton-Raphson's method
4. Numerical solution of the heat equation
5. Sparse Vector Arithmetic

Demonstrations will be done during lab hours.

Matlab function syntax

```
function [y1,...,yN] = myfun(x1,...,xM)
```

declares a function named myfun that accepts inputs x1,...,xM and returns outputs y1,...,yN. This declaration statement must be the **first executable line** of the function.

Save the function code in a text file with a .m extension. The name of the file should match the name of the **first function** in the file.

Valid function names begin with an alphabetic character, and can contain letters, numbers, or underscores.

Files can include multiple local functions or nested functions.

(http://www.mathworks.se/help/matlab/ref/)

Matlab function syntax (cont.)

Use the end keyword to indicate the end of each function in a file if:

– Any function in the file contains a nested function
– Any local function in the file uses the end keyword

Otherwise, the end keyword is optional.

(http://www.mathworks.se/help/matlab/ref/)

Recursion

$$f(n) = \begin{cases} 1 & n = 1 \\ n \times f(n-1) & n > 1 \end{cases}$$

In Matlab:

```
function res = fac1(n)

if n==1
    res = 1;
else
    res = n*fac1(n-1);
end % if

end % fac1
```

Iteration

$$f(n) = \begin{cases} 1 & n = 1 \\ n \times f(n-1) & n > 1 \end{cases}$$

In Matlab:

```
function res = fac3(n)

res = 1;
while n>1
    res = res *n;
    n = n-1;
end % while

end % fac3
```

Stack operations

- createStack: to create a stack
  - precond: None
  - postcond: A stack has been created and initialized to be empty. The stack is returned.
- emptyStack: to check if the stack is empty
  - precond: The stack has been created.
  - postcond: The function returns true (= '1') if it is empty, otherwise false.

createStack and emptyStack

```
function s = createStack;

s = [];

end % createStack



function res = emptyStack(s);

res = (length(s) == 0);

end % emptyStack
```

Stack operations (cont.)

- push
  - precond: The stack has been created and is not full.
  - postcond: The element has been stored as the stack's top element. The updated stack is returned.
- pop
  - precond: The stack has been created and is not empty.
  - postcond: The top element of the stack has been removed and is returned. The updated stack is returned as well.
- top
  - precond: The stack has been created and is not empty.
  - postcond: A copy of the top element of the stack is returned.

push and pop

```
function s = push(el, s);
s = [el s];
end % push


function [el, s] = pop(s);
if emptyStack(s)
    el = [];    disp('error')
elseif length(s) == 1
    el = s(1);
    s = createStack;
else
    el = s(1);
    s = s(2:end);
end % if
end % pop
```

Structure and structure array: example

```
vip.name = 'alice';
vip.day = 3;
vip.month = 4;
vip.year = 1900;


vip(2).name = 'bo';
vip(2).day = 1;
vip(2).month = 12;
vip(2).year = 1950;
```

Manipulate structure array

Store data

```
register(index).field = value
```

is the same as

```
register = setfield(register, {index}, field, value)
```

Retrieve data

```
register(index).field
```

is the same as

```
getfield(register, {index}, field)
```

Search, sequential

```
function data = searchStruct(register, element)

found = 0; index = 1;
len = length(register);
data = [];

while (~found) && (index <= len)
    if element == register(index).day
        found = 1
        data = register(index); %% THIS GOES TO THE OUTPUT
    else
        index = index + 1
    end % if
end % while
end % searchStruct
```

search, seq. cont.

```
function data = searchStruct(register, field, element)

found = 0; index = 1;
len = length(register);
data = [];

while (~found) && (index <= len)
    if element == getfield(register, {index}, field)
        found = 1
        data = register(index); %% THIS GOES TO THE OUTPUT
    else
        index = index + 1
    end % if
end % while
end % searchStruct
```

Binary search

The algorithm finds the position of a specified input value within an array **sorted** by key value.

In each step, it compares the search key value with the key value of the middle element of the array.

```
function data = searchBinStruct(register, field, element)

found = 0;
data = [];
left = 1;
right = length(register);
```

```
while (~found) && (left <= right)
    mid = floor((left + right)/2);
    current = getfield(register, {mid}, field);

    if element < current
        right = mid - 1;
    elseif element > current
        left = mid + 1;
    else
        found = 1;
        data = register(mid); %% THIS GOES TO THE OUTPUT
    end % if
end % while
end % searchBinStruct
```

NB. $\mathit{floor}(x) = \lfloor x \rfloor$ is the largest integer not greater than x