Annotation

- Structures (record data type),
- Binary files,
- Simple sorting algorithms,
- Units.

Record data type

- Usually it happens that our data consist of several values
  (e.g., a point in a plane is parametrized by a pair of numbers).

Record data type

- Usually it happens that our data consist of several values (e.g., a point in a plane is parametrized by a pair of numbers).
- It is not at all handy to keep each value in a different array (it is better to have mutually related values on one pile).

Record data type

- Usually it happens that our data consist of several values (e.g., a point in a plane is parametrized by a pair of numbers).
- It is not at all handy to keep each value in a different array (it is better to have mutually related values on one pile).
- Thus we define a structure. There we can keep related data "together".

Record data type

- Usually it happens that our data consist of several values (e.g., a point in a plane is parametrized by a pair of numbers).
- It is not at all handy to keep each value in a different array (it is better to have mutually related values on one pile).
- Thus we define a structure. There we can keep related data "together".
- We use a keywork record to define a structure.

The record data type definition
examples and syntax

```
type   point=record
              x,y:integer;
       end;
var    a,b:point;
       book:record
              author:  string[100];
              title:   string[100];
              year:   integer;
       end;
```

Accessing the structure elements

We use a binary (infix) operator "." to access a structure:

a.x – member x in a structure (record) a

Individual members behave as variables thus we may read their value, assign a value...

Note that we may pass them as an argument by reference!

The record data type
example

```
var a,b:point; boo:book;
begin
      a.x:=1;
      a.y:=2;
      b.x:=10;
      b.y:=10;
      boo.author:='Mehlhorn, K.';
      boo.title:='Data Structures and Efficient
Algorithms I';
      boo.year:=1984;
end.
```

Array of structures

```
var library:array [1..100] of record
           author:string[25];
           title:string[45];
           year:integer;
    end;
    begin
        for i:=1 to 100 do begin
             readln(library[i].author);
             readln(library[i].title);
             readln(library[i].year);
        end; ...
```

Keyword `with`

While working with structures, it is bothering to name still the
underlying structure whose members we use (e.g.,
a_structure_with_a_long_name_that_we_though_out_after_drinking
Thus we may write:

with structure do statement or block;

and in the statement (or block) we need not to mention the
structure explicitly (we may use them directly).

Construction `with`
an example

```
procedure output(book_with_a_long_name:book);
begin
      with book_with_a_long_name do
            writeln(author:25,name:46,year:5);
end;
...
      for i:=1 to 100 do
            output(library[i]);
...
```

Text and Binary Files
commenting the state of knowledge

- Last time we have learnt about text-files (and it was exactly the same like standard input/output).

Text and Binary Files
commenting the state of knowledge

- Last time we have learnt about text-files (and it was exactly the same like standard input/output).
- We used a variable of type text. It was assigned to a given file, we opened the file, we have read and written and finally we have closed the file.

Text and Binary Files
commenting the state of knowledge

- Last time we have learnt about text-files (and it was exactly the same like standard input/output).
- We used a variable of type text. It was assigned to a given file, we opened the file, we have read and written and finally we have closed the file.
- Text files are nice, but time to time we want to implement, say, a database (e.g., library).

Text and Binary Files
commenting the state of knowledge

- Last time we have learnt about text-files (and it was exactly the same like standard input/output).
- We used a variable of type text. It was assigned to a given file, we opened the file, we have read and written and finally we have closed the file.
- Text files are nice, but time to time we want to implement, say, a database (e.g., library).
- Now we need a file with completely different properties – mainly we want to be able to access (quickly) the kth element (and edit it).

Text and Binary Files
commenting the state of knowledge

- Last time we have learnt about text-files (and it was exactly the same like standard input/output).
- We used a variable of type text. It was assigned to a given file, we opened the file, we have read and written and finally we have closed the file.
- Text files are nice, but time to time we want to implement, say, a database (e.g., library).
- Now we need a file with completely different properties – mainly we want to be able to access (quickly) the kth element (and edit it).
- This is what binary files can do.

Text and Binary Files
commenting the state of knowledge

- Last time we have learnt about text-files (and it was exactly the same like standard input/output).
- We used a variable of type text. It was assigned to a given file, we opened the file, we have read and written and finally we have closed the file.
- Text files are nice, but time to time we want to implement, say, a database (e.g., library).
- Now we need a file with completely different properties – mainly we want to be able to access (quickly) the kth element (and edit it).
- This is what binary files can do.
- We define a file consisting of many elements of a given type (e.g., structure (record)).

Technical background

- Binary files are operated mainly as the text files, just:

Technical background

- Binary files are operated mainly as the text files, just:
- Instead of type `text` we make a variable of type: `file of`
  `desired_type`.
  Example: `var f:file of nonsense;`

Technical background

- Binary files are operated mainly as the text files, just:
- Instead of type `text` we make a variable of type: `file of desired_type`.
  Example: `var f:file of nonsense;`
- Functions `assign`, `reset`, `rewrite`, `read`, `write` and `close` work exactly in the same way, i.e., the first argument is a variable describing the file (i.e., file of ...).

Technical background

- Binary files are operated mainly as the text files, just:
- Instead of type `text` we make a variable of type: `file of desired_type`.
  Example: `var f:file of nonsense;`
- Functions `assign`, `reset`, `rewrite`, `read`, `write` and `close` work exactly in the same way, i.e., the first argument is a variable describing the file (i.e., file of ...).
- Beware of `append`, `readln` and `writeln`.

Technical background

- Binary files are operated mainly as the text files, just:
- Instead of type `text` we make a variable of type: `file of desired_type`.
  Example: `var f:file of nonsense;`
- Functions `assign`, `reset`, `rewrite`, `read`, `write` and `close` work exactly in the same way, i.e., the first argument is a variable describing the file (i.e., file of ...).
- Beware of `append`, `readln` and `writeln`.
- Differences:

Technical background

- Binary files are operated mainly as the text files, just:
- Instead of type `text` we make a variable of type: `file of desired_type`.
  Example: `var f:file of nonsense;`
- Functions `assign`, `reset`, `rewrite`, `read`, `write` and `close` work exactly in the same way, i.e., the first argument is a variable describing the file (i.e., `file of ...`).
- Beware of `append`, `readln` and `writeln`.
- Differences:
  - `filesize` – returns a number of elements (records) in the file

Technical background

- Binary files are operated mainly as the text files, just:
- Instead of type `text` we make a variable of type: `file of desired_type`.
  Example: `var f:file of nonsense;`
- Functions `assign`, `reset`, `rewrite`, `read`, `write` and `close` work exactly in the same way, i.e., the first argument is a variable describing the file (i.e., file of ...).
- Beware of `append`, `readln` and `writeln`.
- Differences:
    - `filesize` – returns a number of elements (records) in the file
    - `seek` – puts a pointer to the given point (size says of how many elements of our type the file consists).

Example
a phone book

```
type entry=record
     name:string[100];
     number:string[20];
end;
var f:file of entry;
```

Inserting…
… into a binary file

```
procedure add;
var ent:entry;
begin readln(ent.name);
      readln(ent.number);
      assign(f,'database.bin');
      {$I-}
      reset(f);
      {$I+}
      if IOResult<>0 then
            rewrite(F);
      seek(f,filesize(f));
      write(f,ent);
      close(f);
end;
```

Writing the file out
i.e., reading the whole file...

```
procedure output;
var i:integer;
    ent:entry;
begin
     assign(f,'database.bin');
     reset(f);
     for i:=1 to filesize(f) do
     begin
          read(f,ent);
          writeln('Name:  ',ent.name,', phone nr: '
,ent.number);
     end;
     close(f);
end;
```

Erasing in a binary file
Function `truncate`

- We may truncate a binary file at a given position using `truncate`.

Erasing in a binary file
Function `truncate`

- We may truncate a binary file at a given position using `truncate`.
- As a parameter we pass the variable of type `file` (binded to a particular file).

Erasing in a binary file
Function `truncate`

- We may truncate a binary file at a given position using `truncate`.
- As a parameter we pass the variable of type `file` (binded to a particular file).
- Example:
  ```
  reset(f);
  truncate(f);
  ```
  erases recent contain completely

Erasing in a binary file
Function truncate

- We may truncate a binary file at a given position using truncate.
- As a parameter we pass the variable of type file (binded to a particular file).
- Example:
  reset(f);
  truncate(f);
  erases recent contain completely
- similarly with rewrite(f);, just for a nonexistent file the program would crash!

Erasing in a binary file
Function `truncate`

- We may truncate a binary file at a given position using `truncate`.
- As a parameter we pass the variable of type `file` (binded to a particular file).
- Example:
  `reset(f);`
  `truncate(f);`
  erases recent contain completely
- similarly with `rewrite(f);`, just for a nonexistent file the program would crash!
- How to erase one entry?

Erasing in a binary file
Function truncate

- We may truncate a binary file at a given position using truncate.
- As a parameter we pass the variable of type file (binded to a particular file).
- Example:
  reset(f);
  truncate(f);
  erases recent contain completely
- similarly with rewrite(f);, just for a nonexistent file the program would crash!
- How to erase one entry?
- Replace it with the last one and truncate (the last one).

Records
back to structures

- Structures can be used for working with (mutually) related data of not necessarily the same type.

Records
back to structures

- Structures can be used for working with (mutually) related data of not necessarily the same type.
- Examples: Library, phone book, accounting records,...

Records
back to structures

- Structures can be used for working with (mutually) related data of not necessarily the same type.
- Examples: Library, phone book, accounting records,...
- Sometimes we want the data to be hegerogeneous.

Records
back to structures

- Structures can be used for working with (mutually) related data of not necessarily the same type.
- Examples: Library, phone book, accounting records,...
- Sometimes we want the data to be hegerogeneous.
- Example: A journal has no author, book has no program committee...

Records
back to structures

- Structures can be used for working with (mutually) related data of not necessarily the same type.
- Examples: Library, phone book, accounting records,...
- Sometimes we want the data to be hegerogeneous.
- Example: A journal has no author, book has no program committee...
- How can be organize a library consisting of books, journals and newspapers?

Records
back to structures

- Structures can be used for working with (mutually) related data of not necessarily the same type.
- Examples: Library, phone book, accounting records,...
- Sometimes we want the data to be hegerogeneous.
- Example: A journal has no author, book has no program committee...
- How can be organize a library consisting of books, journals and newspapers?
- We use so called **variant record.**

Records
back to structures

- Structures can be used for working with (mutually) related data of not necessarily the same type.
- Examples: Library, phone book, accounting records,...
- Sometimes we want the data to be hegerogeneous.
- Example: A journal has no author, book has no program committee...
- How can be organize a library consisting of books, journals and newspapers?
- We use so called **variant record.**
- We define the attributes depending on an indicator variable.

Syntax

- First we define an indicator type (usually enum):
  `type typeofbook=(book,journal,newspaper);`

Syntax

- First we define an indicator type (usually enum):
  ```
  type typeofbook=(book,journal,newspaper);
  ```
- Then we define a structure where we cover this type by the case-clause:
  ```
  type tbook=record
      name:string;
      pagenum:integer;
      case type:typeofbook of
          book:  (author:string);
          journal:  (editorinchief:string;
              color:boolean);
          newspaper:  (neditorinchief:string;
              spam_volume:real;);
      end;
  ```

Example
Library

```
var library:array[1..100] of tbook;
begin
      library[1].name:='Data Structures and
Algorithms I';
      library[1].pagenum:=226;
      library[1].type:=book;
      library[1].author:='Kurt Mehlhorn';

      library[2].name:='40Hex';
      library[2].pagenum:=30;{I guess...}
      library[2].type:=journal;
      library[2].editorinchief:='Darkangel';
      .....
```

Remarks

- Use is (hopefully) clear.

Remarks

- Use is (hopefully) clear.
- Data in the variant part are stored in a *union*, *i.e.*, they are stored one over another! Languages from C-family call it `union`.

Remarks

- Use is (hopefully) clear.
- Data in the variant part are stored in a *union*, *i.e.*, they are stored one over another! Languages from C-family call it `union`.
- We are showing variant record as an ancient ancestor of polymorphism and inheritance that are implemented in object languages.

Remarks

- Use is (hopefully) clear.
- Data in the variant part are stored in a *union*, *i.e.*, they are stored one over another! Languages from C-family call it union.
- We are showing variant record as an ancient ancestor of polymorphism and inheritance that are implemented in object languages.
- The object programming takes place since the beginning of summer term.

Sorting – the motivation

- We have read the data,
- we want to process them in a monotone ordering.
- How to do that? Sort, process.
- Let us consider the data have been read into an array.

The problem of sorting – simple sorting algorithms

- BubbleSort,
- InsertSort,
- SelectSort,
- QuickSort.

Bubblesort

- Geometric interpretation:
  Bubbles in a liquid tend to ascend.

Bubblesort

- Geometric interpretation:
  Bubbles in a liquid tend to ascend.
- The idea: We are comparing pairs of consecutive numbers
  from the first pair to the last one. If they are incorrectly
  ordered, we swap their positions.

Bubblesort

- Geometric interpretation:
  Bubbles in a liquid tend to ascend.
- The idea: We are comparing pairs of consecutive numbers
  from the first pair to the last one. If they are incorrectly
  ordered, we swap their positions.
- Individual elements are "bubbling" in a correct direction.

Bubblesort

- Geometric interpretation:
  Bubbles in a liquid tend to ascend.
- The idea: We are comparing pairs of consecutive numbers
  from the first pair to the last one. If they are incorrectly
  ordered, we swap their positions.
- Individual elements are "bubbling" in a correct direction.
- We iterate this process until no swap takes place.

Bubblesort in pseudocode

- `weswapped:=true;`

- `while weswapped do`
  `begin`
  - `for i:=1 to length - 1 do`
    `begin`
    - `weswapped:=false;`
    - `if numbers[i]>numbers[i+1] then`
      `begin swap(numbers[i],numbers[i+1]);`
      `       weswapped:=true;`
      `end;`
  - `end;`

- `end;`

Complexity of bubble-sort

- How many times we have to iterate the outer (while-)cycle?

Complexity of bubble-sort

- How many times we have to iterate the outer (`while-`)cycle?
- In the i-th iteration the i-th largest element reaches its position!

Complexity of bubble-sort

- How many times we have to iterate the outer (while-)cycle?
- In the i-th iteration the i-th largest element reaches its position!
- Thus it suffices to perform at most n iterations. Complexity of one iteration is also linear ($O(n)$).

Complexity of bubble-sort

- How many times we have to iterate the outer (while-)cycle?
- In the i-th iteration the i-th largest element reaches its position!
- Thus it suffices to perform at most n iterations. Complexity of one iteration is also linear ($O(n)$).
- Thus altogether $O(n^2)$.

Complexity of bubble-sort

- How many times we have to iterate the outer (while-)cycle?
- In the i-th iteration the i-th largest element reaches its position!
- Thus it suffices to perform at most n iterations. Complexity of one iteration is also linear ($O(n)$).
- Thus altogether $O(n^2)$.
- We may implement the algorithm when in odd iterations we bubble from left to right while in even iterations from right to left. This is called **Shakesort**. Its complexity is the same.

Insert- and Select-sort

Selectsort:

- Repeat until the array to sort is empty:
- Find a minimum in the array to sort and add it to the sorted array.

Insertsort:

- Repeat until the array to sort is empty:
- Take the first element of the array to sort and place it onto the correct position in the target array, i.e.:
  find the position where this element should be in the target array, add it there and the rest of the target array move one position further.

Complexity-analysis: We iterate the process n times. One iteration takes at most cn steps (for some constant c). Therefore altogether $O(n^2)$.

Quicksort
sorting using the recursion – the idea

- Sorting one-element-array is trivial (don't do anything, it is already sorted), i.e., just return the input sequence.

Quicksort
sorting using the recursion – the idea

- Sorting one-element-array is trivial (don't do anything, it is already sorted), i.e., just return the input sequence.
- In a nontrivial array A take a pivot p (element that we use for pivoting).

Quicksort
sorting using the recursion – the idea

- Sorting one-element-array is trivial (don't do anything, it is already sorted), i.e., just return the input sequence.
- In a nontrivial array A take a pivot p (element that we use for pivoting).
- Divide the array A into arrays B and C. B consists of the elements smaller than p, C consists of elements larger than p.

Quicksort
sorting using the recursion – the idea

- Sorting one-element-array is trivial (don't do anything, it is already sorted), i.e., just return the input sequence.
- In a nontrivial array A take a pivot p (element that we use for pivoting).
- Divide the array A into arrays B and C. B consists of the elements smaller than p, C consists of elements larger than p.
- Employ recursion on B, employ recursion on C

Quicksort
sorting using the recursion – the idea

- Sorting one-element-array is trivial (don't do anything, it is already sorted), i.e., just return the input sequence.
- In a nontrivial array A take a pivot p (element that we use for pivoting).
- Divide the array A into arrays B and C. B consists of the elements smaller than p, C consists of elements larger than p.
- Employ recursion on B, employ recursion on C
- Output the array B, output pivot p (as many times as it was in A), output C.

Quicksort
complexity analysis

- What's the complexity of the algorithm? How many times we could "employ the recursion"?

Quicksort
complexity analysis

- What's the complexity of the algorithm? How many times we could "employ the recursion"?
- Yes, n-times. If as a pivot we take the minimum, B is trivial and C is one element smaller than A.

Quicksort
complexity analysis

- What's the complexity of the algorithm? How many times we could "employ the recursion"?
- Yes, n-times. If as a pivot we take the minimum, B is trivial and C is one element smaller than A.
- How long takes each "recursion-level"?

Quicksort
complexity analysis

- What's the complexity of the algorithm? How many times we could "employ the recursion"?
- Yes, n-times. If as a pivot we take the minimum, B is trivial and C is one element smaller than A.
- How long takes each "recursion-level"?
- Linearly w. r. t. n (because each element get operated constantly many times).

Quicksort
complexity analysis

- What's the complexity of the algorithm? How many times we could "employ the recursion"?
- Yes, n-times. If as a pivot we take the minimum, B is trivial and C is one element smaller than A.
- How long takes each "recursion-level"?
- Linearly w. r. t. n (because each element get operated constantly many times).
- Altogether, again, $O(n^2)$.

Quicksort
complexity analysis

- What's the complexity of the algorithm? How many times we could "employ the recursion"?
- Yes, n-times. If as a pivot we take the minimum, B is trivial and C is one element smaller than A.
- How long takes each "recursion-level"?
- Linearly w. r. t. n (because each element get operated constantly many times).
- Altogether, again, $O(n^2)$.
- The average-case complexity is $\Theta(n \log n)$ and the algorithm can be improved to gain this complexity by choosing pivot in a smarter way.

Quicksort
complexity analysis

- What's the complexity of the algorithm? How many times we could "employ the recursion"?
- Yes, n-times. If as a pivot we take the minimum, B is trivial and C is one element smaller than A.
- How long takes each "recursion-level"?
- Linearly w. r. t. n (because each element get operated constantly many times).
- Altogether, again, $O(n^2)$.
- The average-case complexity is $\Theta(n \log n)$ and the algorithm can be improved to gain this complexity by choosing pivot in a smarter way.
- To improve this algorithm we want to find a median - but we have to do it in linear time.

FIXME!!!

Here shall be a remark on method "Divide et impera"!
Here should be a quicksort implementation!
Passing a function as an argument.
Odstrasujici priklady (slidy10.tex for mathematicians).

Units
how to compile parts of code separately

■ Sometimes we implement functions usable in several projects
   (e.g., our sorting functions).

Units
how to compile parts of code separately

- Sometimes we implement functions usable in several projects (e.g., our sorting functions).
- We may copy (click'n'paste) them into the other source files (bad idea)

Units
how to compile parts of code separately

- Sometimes we implement functions usable in several projects (e.g., our sorting functions).
- We may copy (click'n'paste) them into the other source files (bad idea)
- or we store them into a separate file that gets compiled separately.

Units
how to compile parts of code separately

- Sometimes we implement functions usable in several projects (e.g., our sorting functions).
- We may copy (click'n'paste) them into the other source files (bad idea)
- or we store them into a separate file that gets compiled separately.
- The latter approach is referred as the **units.**

Units – advantages and disadvantages

- Source code gets spreaded into several files,

Units – advantages and disadvantages

- Source code gets spreaded into several files,
- it is not necessary to write the code more times when we want to share it in several projects.

Units – syntax and semantic

- Instead of word program we start with keyword unit,
- again, we place the name of the unit. This time the name must correspond with the filename. Also the keyword unit is compulsory.
- A unit consists of an interface (what's visible from the outside)
- and of implementation (internal part where the interface is implemented).

Units – the interface part

- Interface describes publicly visible part of the unit.

Units – the interface part

- Interface describes publicly visible part of the unit.
- Interface consists of:

Units – the interface part

- Interface describes publicly visible part of the unit.
- Interface consists of:
- variable definitions (when the variables should be publicly visible),

Units – the interface part

- Interface describes publicly visible part of the unit.
- Interface consists of:
- variable definitions (when the variables should be publicly visible),
- function (and proc.) prototypes (when the function should be publicly visible),

Units – the interface part

- Interface describes publicly visible part of the unit.
- Interface consists of:
- variable definitions (when the variables should be publicly visible),
- function (and proc.) prototypes (when the function should be publicly visible),
- prototype is the header of the function, i.e., the "first line".

Units – impelementation

■ What should *not* be publicly visible, i.e.:

Units – impelementation

- What should *not* be publicly visible, i.e.:
- Function definitions,

Units – impelementation

- What should *not* be publicly visible, i.e.:
- Function definitions,
- variable definitions (for internal variables of the unit),

Units – impelementation

- What should *not* be publicly visible, i.e.:
- Function definitions,
- variable definitions (for internal variables of the unit),
- definition of any stuff that should be (publicly) invisible,

Units – impelementation

- What should *not* be publicly visible, i.e.:
- Function definitions,
- variable definitions (for internal variables of the unit),
- definition of any stuff that should be (publicly) invisible,
- definition of internal functions (not mentioned in interface).

Units – impelementation

- What should *not* be publicly visible, i.e.:
- Function definitions,
- variable definitions (for internal variables of the unit),
- definition of any stuff that should be (publicly) invisible,
- definition of internal functions (not mentioned in interface).
- We finish the unit by keyword end. (followed by full-stop)

Units – example

```
unit sorting;
interface
      type po=array[0..9] of integer;
      procedure bubble(var arr:array of integer);
      procedure select(var a:po);
      procedure insert(var a:po);
      procedure quicksort(var arr:array of
integer;number:integer);
      procedure output(a:array of integer);
```

Units – example (cont.)

```
...
implementation
     var inserted:integer;
     procedure bubble(var arr:array of integer);
     ...
     function extract_min(var a:po):integer;
     {This function will not be visible from
outside!}
     ...
     procedure select(var a:po):integer;
     ...
     ...
end.
```

Units – how to use them

- When using a unit, we announe it with a keyword uses followed by the name of the unit:
- Example: `uses sorting;`

Using the unit – example

```
program sort;
uses sorting;
var p:array [0..9] of integer;
i:integer;
begin
for i:=0 to 9 do
read(p[i]);
quicksort(p,10);
output(p);
end.
```

Standard units

Turbo Pascal is equipped with several standard units:

- crt,
- dos,
- graph,
- printer,
- ...

Units may differ for individual compilers!

Unit `crt`

- Unit operating a keyboard and a display (colors, sounds)
- Variables: `LastMode` (says what textmode was the last one used before switching graphics on),
- `TextAttr` (current attributes for displaying (text). Gets operated by `TextBackground` and `TextColor`),
- Procedure `TextBackground` sets the background color, proc. `TextColor` sets the color of foreground.
- function `keypressed` (returns `boolean` saying whether any key was pressed, `clrscr` (erases the display).

Units dos, graph a printer

- Unit dos works with files, directories, disks...
- Unit graph enables graphic mode (InitGraph, CloseGraph, GraphResult, SetColor, GetColor...).
- Unit Printer serves for printing.
- All these units consist of many functions, procedures and variables. If you want to, you may find them in Help.

Strange example:

Probably you have already several times seens:
```
program nothing;
uses crt;
...
begin
...  repeat until keypressed;
end.
```
What is this?

Strange example:

Probably you have already several times seens:
```
program nothing;
uses crt;
...
begin
...  repeat until keypressed;
end.
```
What is this?
Use of unit crt, namely its function keypressed.

Directive `forward`

- It is typical that one function calles another but

Directive `forward`

- It is typical that one function calles another but
- sometimes the latter function calls the former, too.

Directive `forward`

- It is typical that one function calles another but
- sometimes the latter function calls the former, too.
- Problem: In Pascal we have to define first (then we may use).

Directive `forward`

- It is typical that one function calles another but
- sometimes the latter function calls the former, too.
- Problem: In Pascal we have to define first (then we may use).
- Cyclic dependence seems unsolvable...

Directive forward

- It is typical that one function calles another but
- sometimes the latter function calls the former, too.
- Problem: In Pascal we have to define first (then we may use).
- Cyclic dependence seems unsolvable...
- until we find the forward directive!

Directive `forward`

- It is typical that one function calles another but
- sometimes the latter function calls the former, too.
- Problem: In Pascal we have to define first (then we may use).
- Cyclic dependence seems unsolvable...
- until we find the `forward` directive!
- This directive is placed after the function prototype:

Directive forward

- It is typical that one function calles another but
- sometimes the latter function calls the former, too.
- Problem: In Pascal we have to define first (then we may use).
- Cyclic dependence seems unsolvable...
- until we find the forward directive!
- This directive is placed after the function prototype:
- procedure two(a:integer);forward;

Forward example:

```
program qq;
procedure two(a:integer);forward;
procedure one(a:integer);
begin
      two(a);
end;
procedure two(a:integer);
begin
      one(a);
end;
begin
      one(1);
      {Let us ignore that this program does
not make a good sense!}
end
```