


Eötvös Loránd University
Faculty of Informatics

PROGRAMMING

2019/20 1st semester

Lecture 5

ZS. PLUHÁR, H. TORMA

Content

- Definition of type
- Text vs. array – comparison + tasks with text
- Compund types – overview
- Functions – algorithmic abstraction

The definition of type
A short summary

The Type

› Value set:

Record type:
Type TR=**Record(**
fn_1:TF_1,
…
fn_N:TF_N**)**

› Operators:

- • := •
- •.fn_1
- …
- •.fn_N

The definition of type
A short summary

Record type:

Type TR=Record(

fn_1:TF_1,

...

fn_N:TF_N)

Operators:

- • := •
- •.fn_1

...

- •.fn_N

C++ structure type

```
typedef struct {
    TF1 fn1;
    ...
    TFN fnN ; } TR;
```

- • = •
- •.fn_1

. . .

- •.fn_N

The definition of type
A short summary

The Type

› Value set:

Array type:
Type TA=**Array** [Tindex:Titem]

› Operators:

• := •
•[•]

The definition of type
A short summary

Array type:

Type TA=Array [
Tindex:Titem]

Operators:

- := •
- •[•]

C++ array type

```
typedef Titem [
Cardinality'Tindex];
```

- • = •
- •[• -Min'Tindex]

The definition of type example

We have birth dates from N people, let's find who has birthday in the year first!

Specification

Input: $N \in \mathbb{N}$,

$D_{1..N} \in (month \times day)^N$, $month, day = \mathbb{N}$

Output: $First \in \mathbb{N}$

Precondition: $N > 0$ and

$\forall i (1 \leq i \leq N) : D_i.month \in [1..12]$ and

$D_i.day \in [1..31]$

Postcondition: $1 \leq First \leq N$ and

$\forall i (1 \leq i \leq N)$: $D_{First}.month < D_i.month$ **or**

$(D_{First}.month = D_i.month$ **and** $D_{First}.day \leq D_i.day)$

The definition of type example

We have birth dates from N people, let's find who has birthday in the year first!

Specification

Input: $N \in \mathbb{N}$,

$D_{1..N} \in Date^N$, $Date = month \times day$,
$month, day = \mathbb{N}$

...

Definition:

$\leq : Date \times Date \rightarrow \mathbb{L}$

$d1 \leq d2 \leftrightarrow d1.month < d2.mont \text{ or } d1.month = d2.month \text{ and } d1.day \leq d2.day$

The definition of type example

We have birth dates from N people, let's find who has birthday in the year first!

Algorithm

Max_Select

Max:=1	
i=2..N	
T: X[i]>X[Max]	F: -
Max:=i	

FirstBirthday

First:=1	
i=2..N	
T: D[i]<D[First]	F: -
First:=i	

Input: $N \in \mathbb{N}$, $X_{1..N} \in \mathbb{S}^N$

Output: $Max \in \mathbb{N}$

Precondition: $N > 0$

Postcondition: $1 \leq Max \leq N$ and $\forall i\,(1 \leq i \leq N) : X_{Max} \geq X_i$

The < operator to type Date needs to be defined!

Text type

Similarities between text and array

› consists of elements of the same type,

› can be indexed;

Differences:

› the array is parameterized with the type of its elements (and index), whilst text always consists of characters,

› in an algorithm, text index always starts by 1, but array start index depends on declaration,

› array length is constant, but text length can vary; in the case of text, we have `length()` function and `+` operator

› modifying an element is problematic with text type

Text type in C++

Most important string operations (`string a;`):

```
…""…                    // empty text; e.g.: a="";
cin >> a;               // read until space or endline
getline(cin,a,'\n');    // read until '\n'
getline(cin,a,'x');     // read until 'x' char
…a.length()…            // number of characters
…a.size()…              // the length in bytes
…+…                     // concatenate
…a[i]…                  // ith character, 0≤i<a.length()
…a.at(i)…               //=a[i], by object
                        // notation, it is checked
```

$a+t \equiv a_1+\ldots+a_{length(a)}+t_1+\ldots+t_{length(t)}$
$\rightarrow length(a+t)=length(a)+length(t)$

Text type in C++

Most important string operations (string a;):

```
a.find(what)…                 // location of what in a
a.substr(from,cnt)…           // =a[from..from+cnt-1]
a.replace(from,cnt,what); // replace 'cnt' chars
                              // substring starting at
                              // 'from' with 'what'
```

Related character operations (string a;)

```
isalpha(a[i])…                // a[i] ’a’..’z’, ’A’..’Z’?
isdigit(a[i])…                // a[i] ’0’..’9’?
isupper(a[i])…                // a[i] ’A’..’Z’?
islower(a[i])…                // a[i] ’a’..’z’?
tolower(a[i])…                // converts a[i] to lower case
toupper(a[i])…                // converts a[i] to upper case
```

Tasks with texts

Let's reverse the order of letters in a word (text)!

Specification (copy PoA):

Input: $O \in \mathbb{T}$

Output: $R \in \mathbb{T}$

Precondition: –

Postcondition: $length(R)=length(O)$ and

$\forall i(1 \leq i \leq length(O))$: $R_i = O_{length(O)-i+1}$

Pre-defined function:

$length: \mathbb{T} \rightarrow \mathbb{N}$

length(t): character count of t

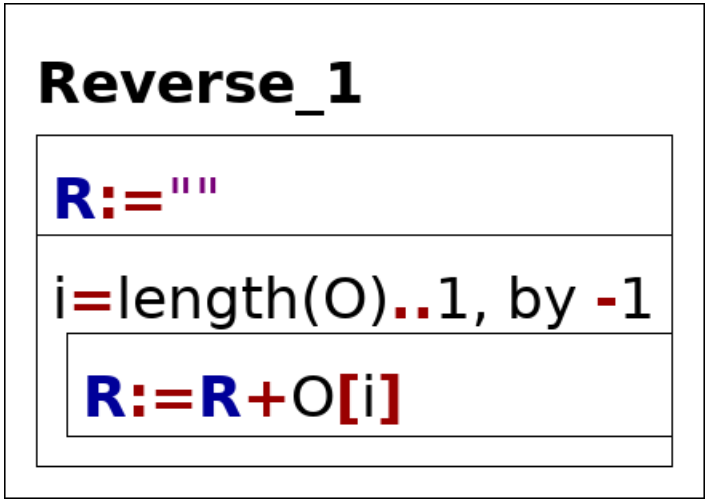
Tasks with texts

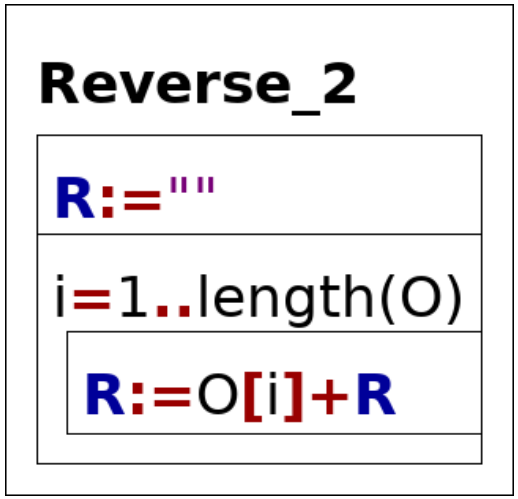
Let's reverse the order of letters in a word (text)!

Input: $O \in \mathbb{T}$
Output: $R \in \mathbb{T}$
Precondition: –
Postcondition: $length(R)=length(O)$ and $\forall i (1 \leq i \leq length(O)): R_i = O_{length(O)-i+1}$

Algorithm (sequence calculations PoA!):

› We cannot modify the i^{th} character of a text, if it does not exist. We need the + operator.





Tasks with texts

Let's define the initials of an English name (e.g. Joe Black → JB)!

Specification:

Input: $\texttt{Name} \in \mathbb{T}$

Output: $\texttt{Ini} \in \mathbb{T}$

Precondition: `isCorrect(Name)`

Comment: the name is correct, if all the initial letters are capital letters, but just the initials, nothing else. We assume there exists function:

$$\texttt{IsCorrect}: \mathbb{T} \to \mathbb{L}$$

Tasks with texts

Let's define the initials of an English name (e.g. Joe Black → JB)!

Specification (cont.):

Postcondition: $\texttt{length(Ini)} = \sum_{\substack{i=1 \\ \texttt{IsCapital}(Name_i)}}^{\texttt{length(Name)}} 1$

$\texttt{and } \forall \texttt{i}(1 \leq \texttt{i} \leq \texttt{length(Ini)}):$
$\texttt{IsCapital}(\texttt{Ini}_\texttt{i}) \texttt{ and Ini} \subseteq \texttt{Name}$

another way: $\texttt{Ini} = \mathit{multiselect}_{\substack{i=1 \\ IsCapital(Name_i)}}^{length(Name)} Name_i$

Comment:

› there exists function „$\texttt{IsCapital}:\mathbb{C}\texttt{h} \rightarrow \mathbb{L}$” (char → boolean)

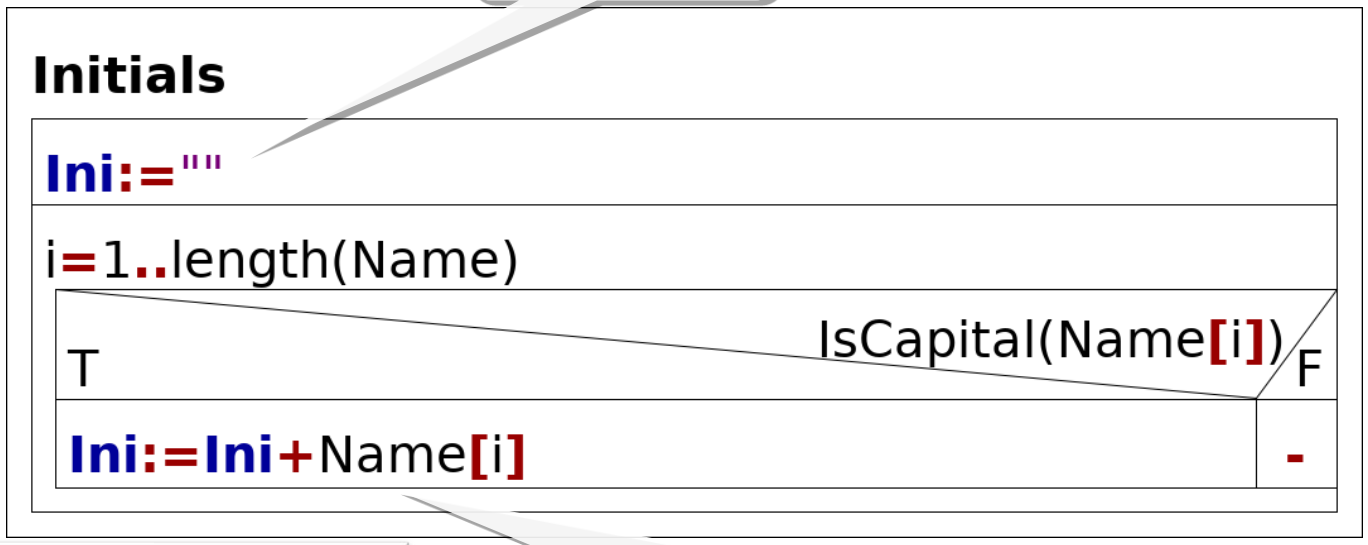
› „$\subseteq$” operator, is the subsequence operator ($x \subseteq y$ is true, if we can get x by eliminating certain elements of y).

Tasks with texts

Let's define the initials of an English name (e.g. Joe Black → JB)!

Algorithm:

length(Ini)=0



Postcondition: $\texttt{length(Ini)}=\sum_{i=1}^{\texttt{length(Name)}}{}_{\texttt{IsCapital(Name}_i\texttt{)}}\ 1$

and $\forall i(1\le i\le \texttt{length(Ini)})$: $\texttt{IsCapital(Ini}_i\texttt{)}$ and $\texttt{Ini} \subseteq \texttt{Name}$

$\texttt{Ini+Name}_i \equiv \texttt{Ini}_1+\ldots+\texttt{Ini}_{\texttt{length(Ini)}}+\texttt{Name}_i$

$\rightarrow \texttt{length(Ini+Name}_i\texttt{)=length(ini)+1}$

Tasks with texts

Let's define the initials of a Hungarian name (e.g. Szabó Éva ⇒ SzÉ)!

Specification:

Input: `Name`$\in\mathbb{T}$

Output: `Ini`$\in\mathbb{T}$

Precondition: `isCorrect(Name)`

Postcondition: homework

Problem: in Hungarian we have double letters, where the second one is not a capital in the initial

Tasks with texts

Let's define the initials of a Hungarian name (e.g. Szabó Éva ⇒ SzÉ)!

Idea for solution:

› Representing data:

Each double letter start with capital letter

```
DoubleL∈𝕋⁸ =
("Cs","Dz","Gy","Ly","Ny","Sz","Ty","Zs")
```

Question: How would you deal with „Dzs”?

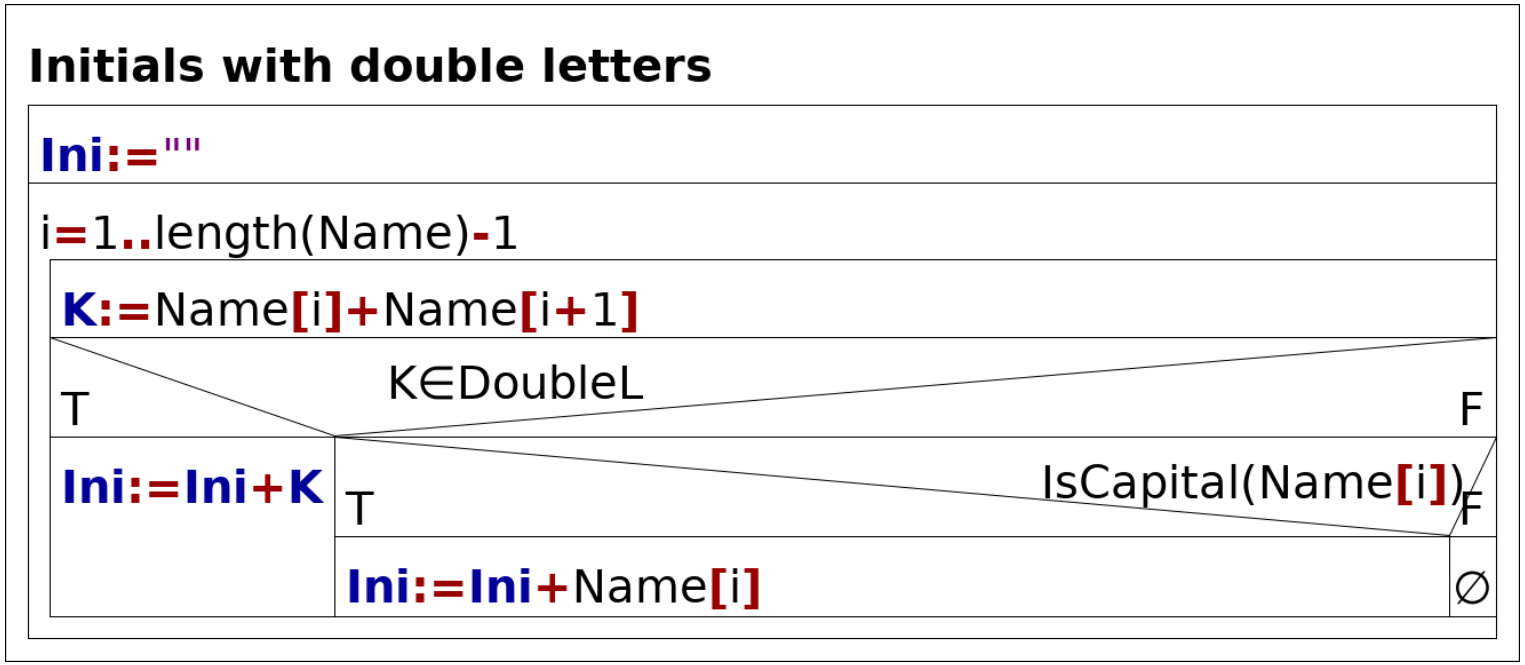
› A draft for the **algorithm**:

Let's define all two-character parts of Name, then check which one is part of `DoubleL` array, or starts with a capital letter

Tasks with texts

Let's define the initials of a Hungarian name (e.g. Szabó Éva ⇒ SzÉ)!

Algorithm:

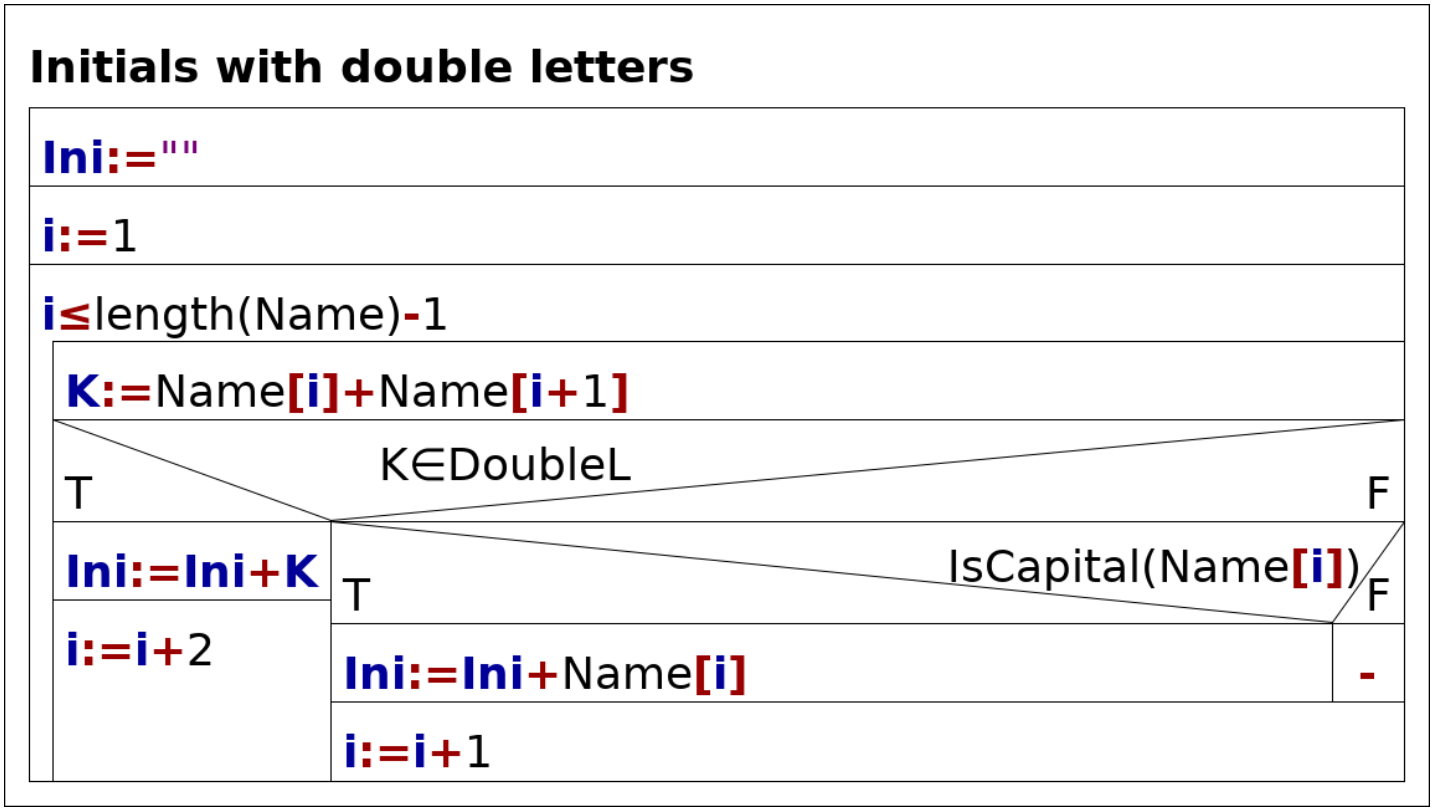


1) Can a name be „Nagy A”? 2) Can we change the order of conditional statements? 3) Is it an optimal solution?

Tasks with texts

Let's define the initials of a Hungarian name (e.g. Szabó Éva ⇒ SzÉ)!

Algorithm:



Tasks with texts

We have two letters, let's decide which one precedes the other in the English alphabetical order!

Specification:

Input: $A, B \in \mathbb{C}h$

Output: $APrecedes \in \mathbb{L}$

Precondition: `isLetter(A), isLetter(B)`

Postcondition: $APrecedes = A <_E B$

Definition: $x <_E y$ if and only if ???

Assumption:

there exists function $IsLetter : \mathbb{C}h \rightarrow \mathbb{L}$

Tasks with texts

We have two letters, let's decide which one precedes the other in the English alphabetical order!

Idea for the solution:

› Let's store all the characters in the right order, and if we find A precedes B in this array, then output is true → use Selection PoA twice

Definition:

`Letters∈ℂh^(2*26)=("a","A","b","B",…,"z","Z")`

$x <_E y$ if and only if $i<j$: $x=Letters_i$ and $y=Letters_j$

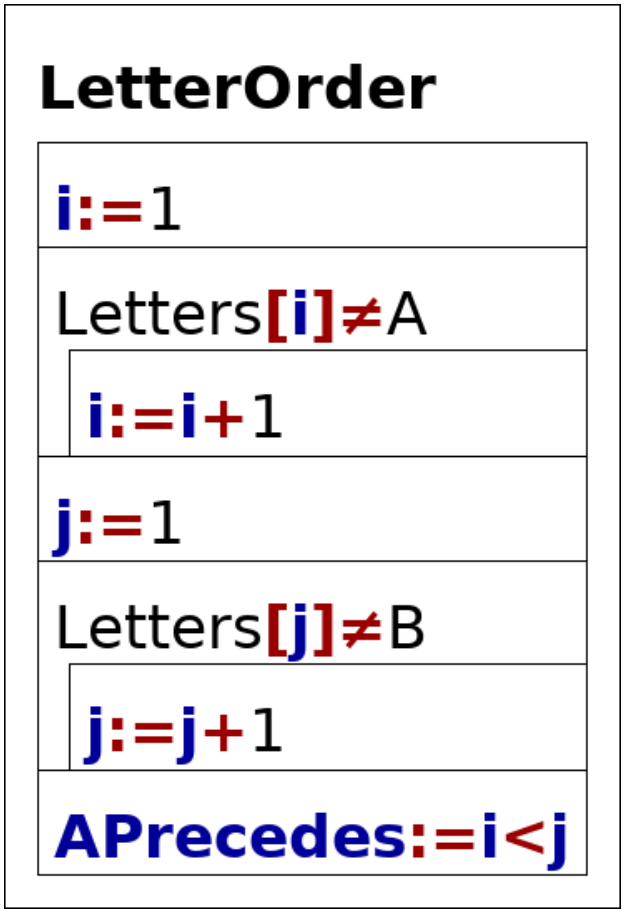
Another solution based on text type:

`Letters∈𝕋="aAbB…zZ"`

Tasks with texts

We have two letters, let's decide which one precedes the other in the English alphabetical order!

Algorithm:

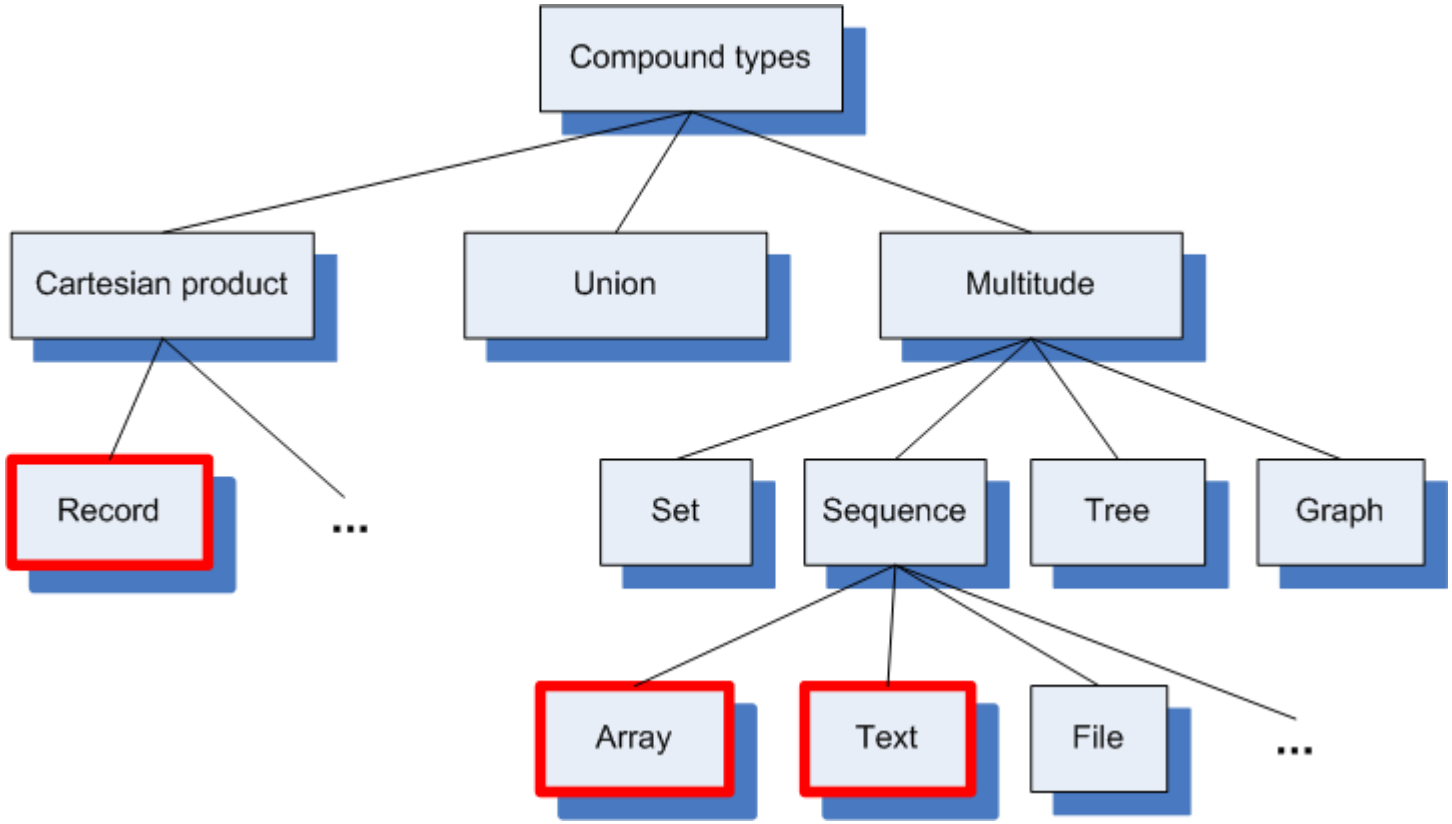


Questions:

- What if the precondition was not met?
- Could be "a"="A"?

```
(i-1) div 2 < (j-1) div 2
```

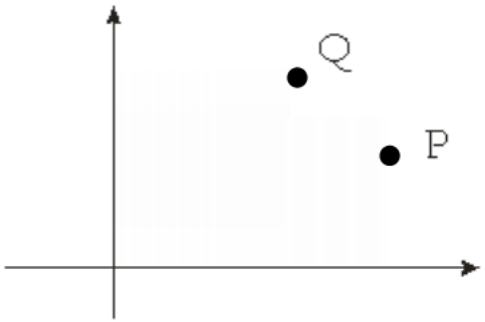
Compound types



Functions - directions

Task: Let's determine if – viewing from the origin – point Q can be seen to the left or to the right relative to point P which is located in the first quarter-plane!

$$\text{Direction}(P,Q) = \begin{cases} -1, \text{if to the left} \\ +1, \text{if to the right} \\ 0, \text{if same direction} \end{cases}$$

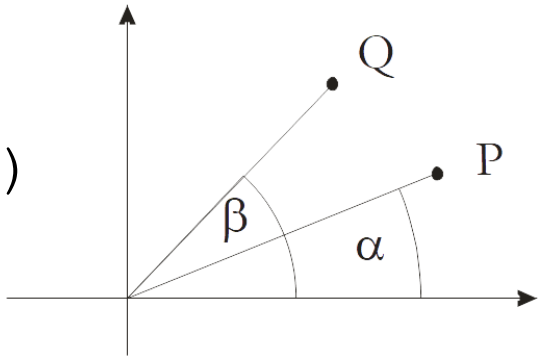


Functions - directions

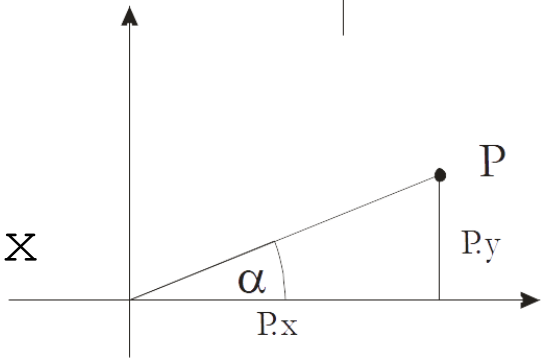
Interpretation:

A point's „direction" can be defined by its azimuthal angle (the angle between its vector and axis x).

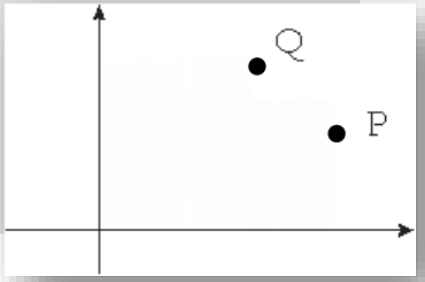
$\alpha < \beta \rightarrow \tan(\alpha) < \tan(\beta)$



$\tan(\alpha) = P.y / P.x$



Functions - directions



$\alpha<\beta \leftrightarrow$ `tan(`α`)<tan(`β`)` $\leftrightarrow$

`P.y/P.x<Q.y/Q.x` $\rightarrow$

`P.y*Q.x<Q.y*P.x` $\leftrightarrow$

`P.y*Q.x-Q.y*P.x<0`

Predicate:

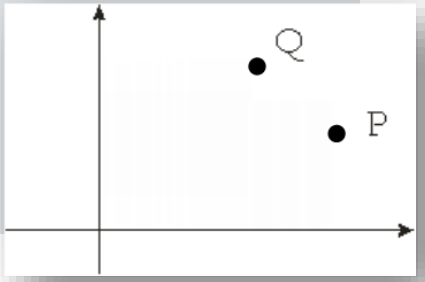
`Direction(P,Q)=sgn(P.y*Q.x-Q.y*P.x)`

(and it is true in all quadrants).

checking if it's true:

`sgn(P.y*Q.x-Q.y*P.x)` $= \begin{cases} -1, \text{if Q is left to P} \\ +1, \text{if Q is right to P} \\ 0, \text{if same direction} \end{cases}$

Functions - directions



Specification:

Input: $P,Q \in Point,\ Point\ x \times y,\ x,y=\mathbb{Z}$

Output: $Dir \in \mathbb{Z}$

Precondition: –

Postcondition: `Dir=Direction(P,Q)`

Actual parameters

Function notation:
`Fn: from → to`

Definition: $Direction: Point \times Point \rightarrow \mathbb{Z}$

`Direction(p,q)=sgn(p.y*q.x-q.y*p.x)`

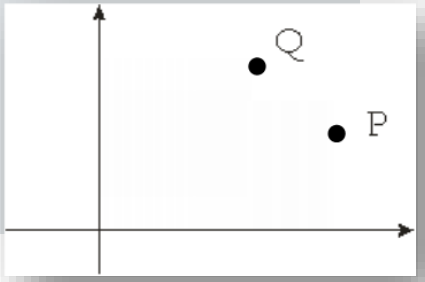
Formal parameters

Algorithm:

```
Dir:=Direction(P,Q)
```

Actual parameters

Functions - directions



- Let's take the function Direction as a separate task.
- Specification:

  Input: $P,Q \in Point$, $Point\ x \times y$, $x,y = \mathbb{Z}$

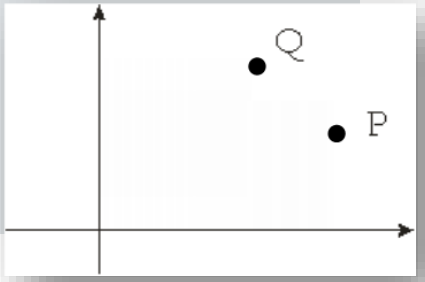
  Output: $Direction(P,Q) \in \mathbb{Z}$

  Precondition: –

  Postcondition: $Direction(p,q) = sgn(p.y*q.x - q.y*p.x)$

- In the input, we can find the function parameters ($\in$ *function domain*), in the output, the parameterized value of the function ($\in$ *function range*), in the postcondition, the connection.

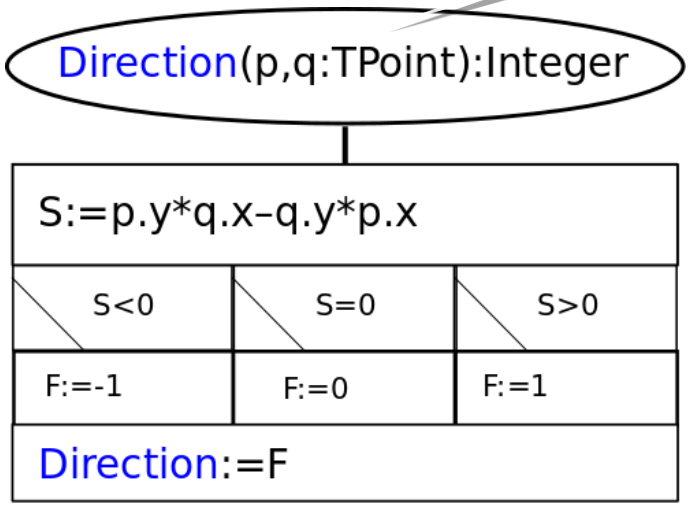
Functions - directions

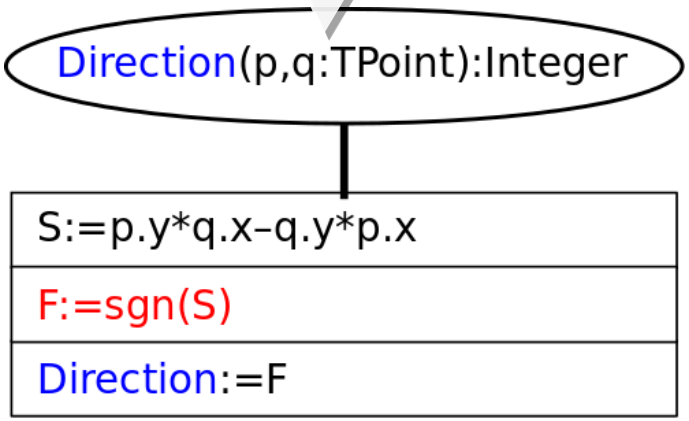


specification → algorithm

Function definition

Formal parameters





Concepts in C++

Block: The program text between `{` and the belonging closing `}`.

```
int main()
{
   int i=1,j=2;
   for (int i=1;…){
      cout<<i…;
      j++;
      …
   }
   cout<<i+j…;
   …
}
```

Concepts in C++

Scope: The scope of an X identifier is the part of the program text (not necessarily contiguous) where one can refer to X.

- Scope is defined by the structure of blocks. If two blocks have common parts, then one of those entirely contains the other.
- The scope of an identifier reaches from the following character of the declarion up to the closing } of the block, except for the blocks (and it's descendants) where it was re-declared.

Concepts in C++

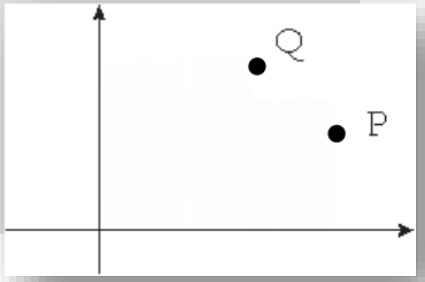
› **Scope**:

An identifier that is referred to at a certain place is **local**, if it was declared in the smallest block containing the reference. An identifier is **global** (for the given block) if it is not local.

› **Lifetime**:

The lifetime of all the variables declared in block B lasts from entering the block to the completion of the last instruction of the block.

Functions - directions

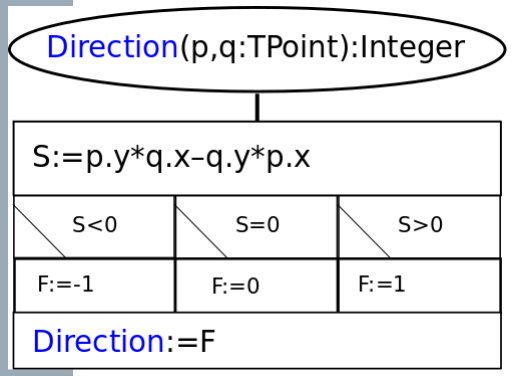


algorithm → code

Function call in the main program:

```
Dir=Direction(P,Q);
```

Definition of the function:




```
int Direction(TPoint p, TPoint q){
 int S; int F; //aux variables
 S=p.y*q.x-q.y*p.x;
 if (S<0) F=-1;
  else if (S==0) F=0;
    else if (S>0) F=1;
 return F;
}
```

Functions – C++ summary

Function head declaration (prototype):

```
funcType funcId(parType formParId,…);
        or
void funcId(parType formParId,…);
```

If there are no formal parameters, the parentheses are still needed

Definition of the function:

The function head without the semicolon

```
{
… //body of function
  return funcValue;
}
```

in the case of void, return without funcValue, or without return


Functions – C++ summary

Formal scalar parameter:

input → no special keyword

output → & is the special prefix keyword

Actual scalar parameter:

if the belonging formal parameter is

input → it can be constant or variable

output → can only be variable

Functions – C++ summary

Passing scalar parameters:

- **By value** – a local identifier is created from the formal parameter, and the value of the actual parameter is copied into it. Thus changing the parameter within the function body won't effect the actual parameter. E.g.:

```
int max(int x, int y);
```

- **By reference** – the address (memory reference) of the actual parameter is copied into the formal parameter, so it will be available through its local name.

```
void max(int x, int y, int &maxxy);
```

Functions – C++ summary

Formal array parameter:

input → **const** prefix keyword

output → no special keyword

Actual array parameter:

if the belonging formal parameter is

input → it can be constant or variable

output → can only be **variable**

Functions – more details regarding C++ code

Array as parameter:

› Principle: arrays are always passed **by reference** as a parameter

› Input parameter – function head example:

Note: does not work with matrices

```
void out_int_array(const int x[], int n);
                        OR
void out_int_array(const int x[maxN], int n);
```

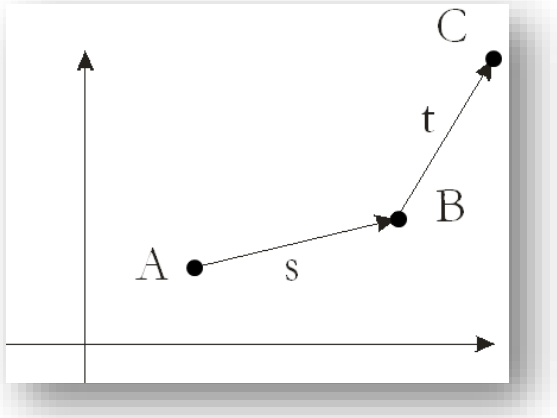
› Output parameter – function head example:

```
void in_int_array(const int x[], int &n, int maxN);
                        OR
void in_int_array(const int x[maxN],int &n, int maxN);
```

With the purpose of checking input parameters

Functions - turning

Task: In which direction does segment **s** (A→B) turn related to segment **t** (B→C)?

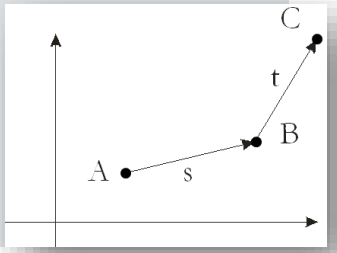


Idea for solution:

Let's shift segments **s** and **t** so that point A gets to the origin. With this transformation, the task became similar to the "Direction" task.

```
Turn(A,B,C)=Direction(B-A,C-A)
```

Functions - turning



Specification:

Input: $A,B,C \in Point,\ Point\ x \times y,\ x,y=\mathbb{Z}$

Output: $TurnD \in \mathbb{Z}$

Precondition: –

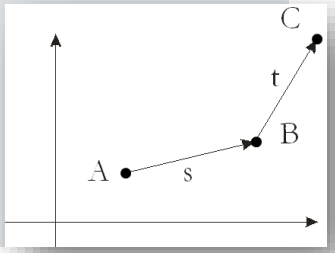
Postcondition: $TurnD=Turn(A,B,C)$

Definition: $Turn:Point^3 \rightarrow \mathbb{Z}$

$Turn(a,b,c):=Direction(b-a,c-a)$

Note: An equivalent task to this: Is point C left or right to the line going through (A, B), or is it on the line going through (A, B)?

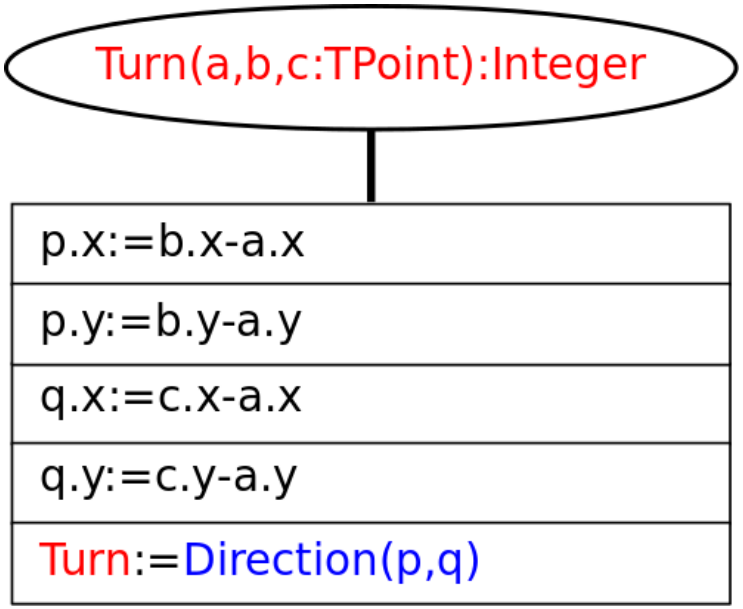
Functions - turning



Algorithm:

```
TurnD=Turn(a,b,c)
```

We call function `Direction` in the solution.




```
int Turn(TPoint a, TPoint b,TPoint c){
   TPoint p,q;
   p.x=b.x - a.x;
   p.y=b.y - a.y;
   q.x=c.x - a.x;
   q.y=c.y - a.y;
  return Direction(p,q);
}
```

Functions – on the segment

Task: Let's decide whether point C is on the segment (A,B)!

Specification:

Input: `A,B,C∈Point, Point=(x×y), x,y=ℤ`

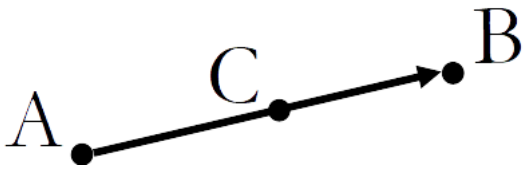
Output: `onSegment∈𝕃`

Precondition: –

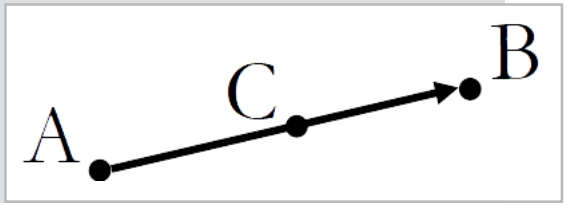
Postcondition: `onSegment=On(A,B,C)`

Definition: `On:Point³→𝕃`

`On(a,b,c):=…`



Functions – on the segment



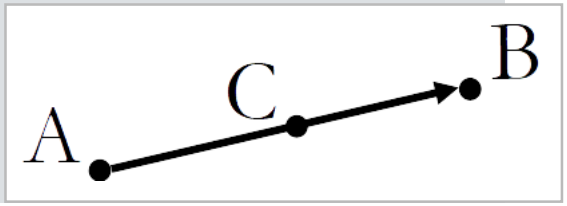
Definition (which consists of two function definitions – postconditions):

```
On(a,b,c):= Turn(a,c,b)=0 and
Between(a.x,c.x,b.x) and
Between(a.y,c.y,b.y)
```

So we need to define another function which decides whether the second parameter is ***between*** the two others

```
Between:ℤ³→𝕃
Between(r,s,t):= r ≤ s ≤ t or t ≤ s ≤r
```

Functions – on the segment



Algorithm: `onSegent:=on(A,B,C);`

More precise:

On(a,b,c:TPoint):Logical

On:= Turn(a,c,b)=0 and
Between(a.x,c.x,b.x and
Between(a.y,c.y,b.y)

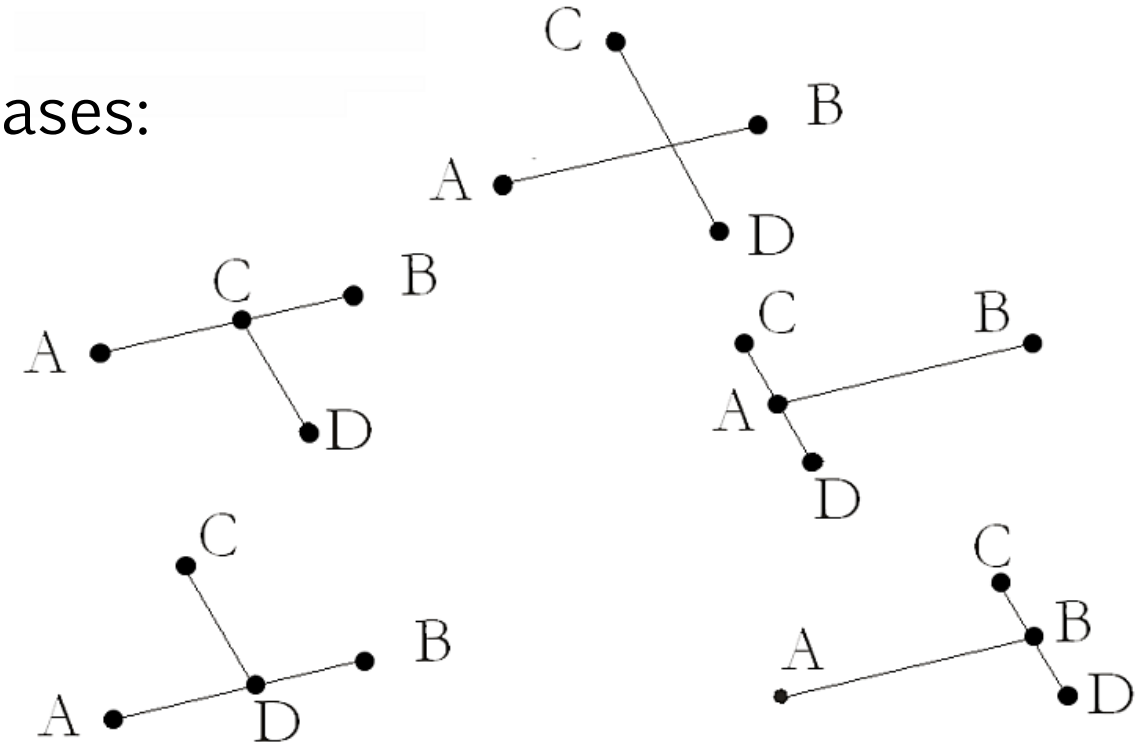
Between(r,s,t:Integer):Logical

Between:= r ≤ s and s ≤ t or t ≤ s and s ≤ r

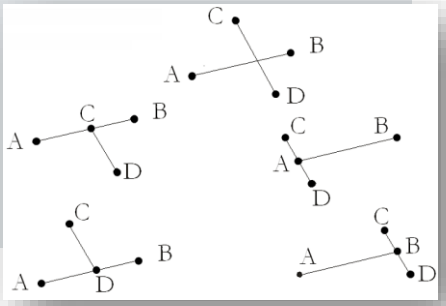
Functions – intersects

Task: Let's decide whether segment (A,B) intersects segment (C,D)

Possible cases:



Functions – intersects



Specification:

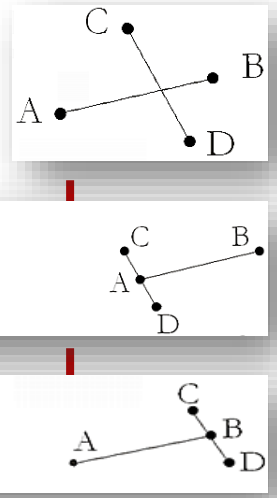
Input: $A,B,C,D \in Point,\ Point\ x \times y,\ x,y=\mathbb{Z}$

Output: $InterS \in \mathbb{L}$

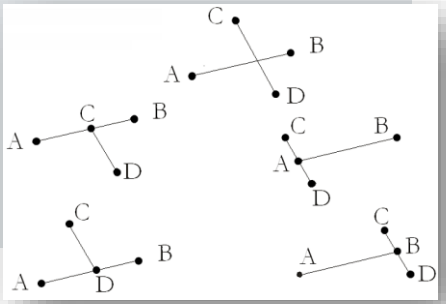
Precondition: $A \neq B$ and $C \neq D$

Postcondition: $InterS=Intersects(A,B,C,D)$

(Turn(A,B,C)*Turn(A,B,D)<0 and
Turn(C,D,A)*Turn(C,D,B)<0
or
On(A,B,C) or On(C,D,A)
or
On(A,B,D) or On(C,D,B))



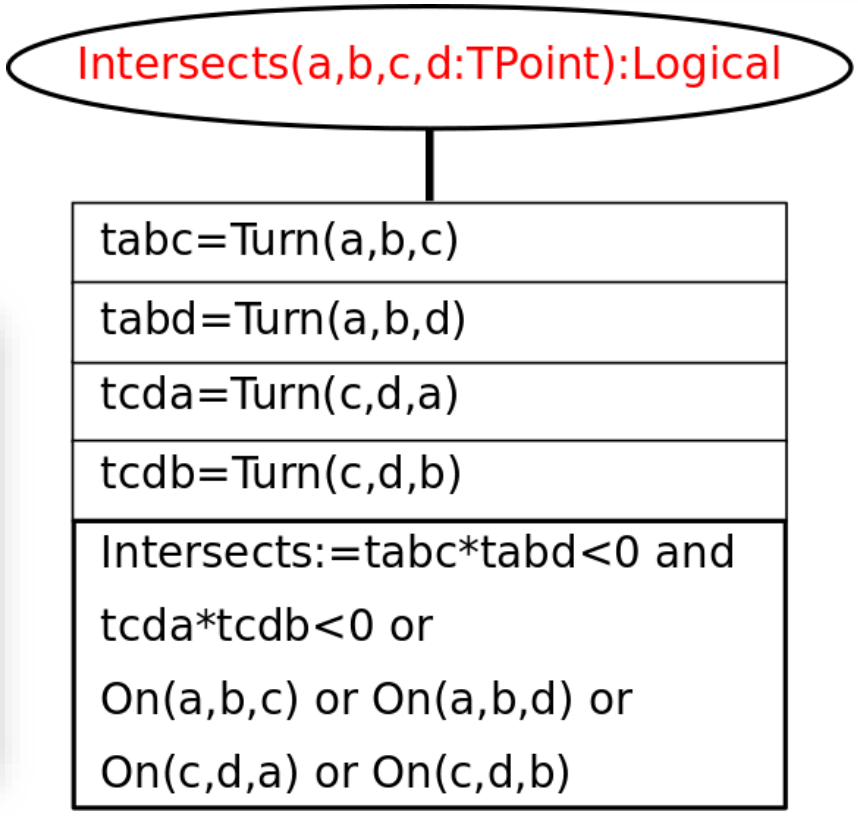
Functions – intersects



Algorithm:

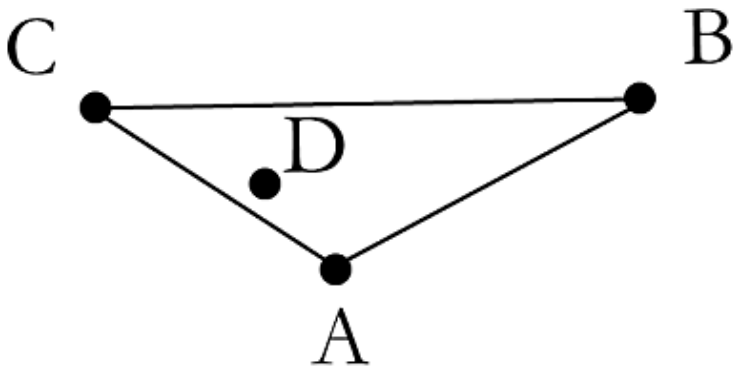
Input: A,B,C∈Point, Point x×y, x,y=ℤ
Output: InterS∈𝕃
Precondition: A≠B and C≠D
Postcondition: InterS=Intersects(A,B,C,D)

```
( Turn(A,B,C)*Turn(A,B,D)<0 and
Turn(C,D,A)*Turn(C,D,B)<0
or On(A,B,C) or On(C,D,A)
or On(A,B,D) or On(C,D,B) )
```



Functions – in triangle

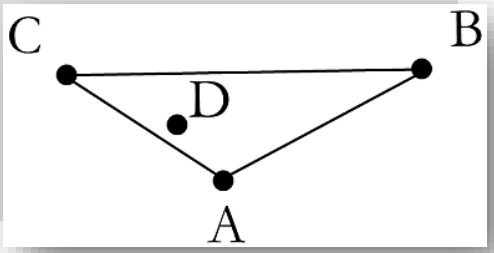
Task: Let's decide whether point D is inside triangle (A,B,C) or not.



Idea for the solution:

Point D is inside if moving around the triangle (A→B→C→A) point D is always on the left hand side or always on the right hand side.

Functions – in triangle



Specification:

Input: $A,B,C \in Point,\ Point=(x \times y),\ x,y=\mathbb{Z}$

Output: $Inside \in \mathbb{L}$

Precondition: $A \neq B \text{ and } B \neq C \text{ and } A \neq C$

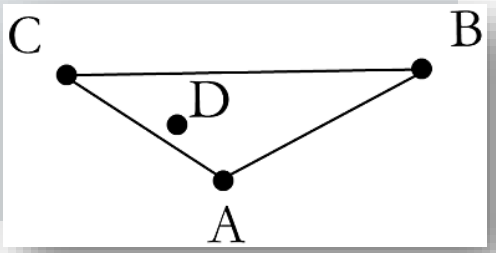
Postcondition:

$Inside=InTriangle(A,B,C,D)$

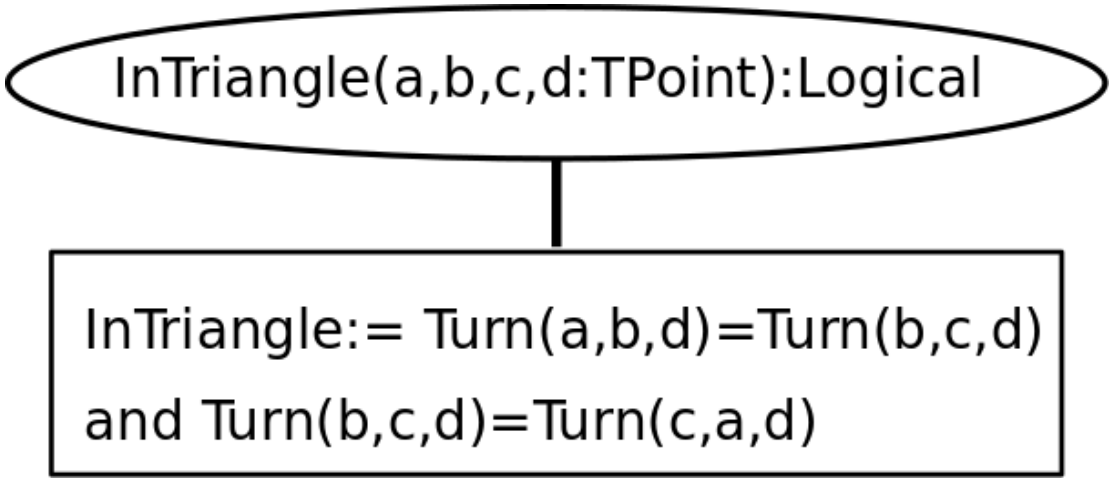
Definition: $InTriangle:Point^4 \rightarrow \mathbb{L}$

```
InTriangle(a,b,c,d):=(
Turn(a,b,d)=Turn(b,c,d) and
Turn(b,c,d)=Turn(c,a,d)-a) )
```

Functions – in triangle



Algorithm:



Functions

Functions built upon each other:

The macrostructure of our functions