

Consider the following declarations:

```
type T1 is array(1..10) of integer;  
type T2 is array(1..10) of integer;  
type T3 is record A: integer; B: Integer; end record;  
type T4 is record A: integer; B: Integer; end record;  
type T5 is array(1..10) of T1;
```

```
A,B: T1;  
C: T2;  
D: T3;  
E: T4;  
F: T5;  
G: array(1..10) of integer;  
H,I: array(1..10) of integer;  
J: T4;  
K: record A: integer; B: Integer; end record;
```

```
function Y(L: T1) return T2;  
function Z(M: array(integer range <>) of integer) return T2;  
function Z(N: T2) return T2;
```

Which of the variables, parameters and function results are assignable to each other, if the rule for assignment requires type equivalence based on

- a) name equivalence
- b) structural equivalence

=====
Consider the following program fragment:

```
C, X: Integer;
```

```
procedure test (A: in out Integer) is
```

```
  I: Integer;
```

```
  type Arr is array(1..2, 1..C, 1..A) of Integer;
```

```
  ArrObj: Arr;
```

```
  K: Integer;
```

```
begin
```

```
  C:= 17;
```

```
  A:= 20;
```

```
  ... ArrObj(2,2,3)... -- (1)
```

```
end test;
```

```
... C:= 2; X:= 5; test(X);
```

Assume that arrays are stored in row-major order.

Draw the structure of the activation record and its contents for this call of test at the point (1), inclusive of any administrative data needed. you are not allowed to use the heap.

Draw the internal structure of the array ArrObj.

Determine the address computation needed to access the array component at point (1).

=====
Consider the following program:

```
I: Integer;  
A: array(1..3) of integer := (0,0,0);
```

```
procedure test(X: Integer) is  
begin  
  A(2) := 7;  
  I := 3;  
  X := X + 1;  
end test;
```

```
begin  
  I := 2;  
  test(A(I));  
  print(A);  
  print(I);  
end;
```

What is the output of this program when parameter-passing is done by

- call by value
- call by value-result
- call by reference
- call by name (ALGOL-Regel)
- call by name (originale Namensersetzungsregel)

The procedure test is now implemented by

```
procedure test(X: Integer) is
```

```
I: Integer := 1;
```

```
begin
```

```
  A(2) := 7;
```

```
  X := X + 1;
```

```
end test;
```

Does this alter your answers and if so in which ways and why ?

=====

We talked about aliasing of objects, when the identifiers designate identical or overlapping storage places.

Identify two programming situations involving parameters in which aliasing can affect the results of a program.

Which other programming constructs can cause aliasing effects ?

What are the problems that aliasing causes from the programmer's point of view ?

What style guidelines can you provide to avoid aliasing ?

=====
Consider the following C-program:

```
#include "stdio.h"

char vek[6]="AEIOU";
char *ptr;

void print (char c)
{ putchar(c); putchar ('\n'); }

void main()
{ ptr=vek;
  print( *ptr );
  print( *ptr+1 );
  print( *ptr );
  print( *(ptr+1) );
  print( *ptr );
  print( *ptr++ );
  print( *ptr );
  print( *(ptr++) );
  print( *ptr );
  print( *++ptr );
  print( *ptr );
  print( ++*ptr );
  print( *ptr );
}
```

Which output is created the calls on print. Be careful about the operator precedence. You may have to consult the C Reference Manual to be sure.

=====

Which property is necessary for function f in the following example, so that that printed result is UNEQUAL.

```
IF f(x) <> f(x) THEN
  WriteString ("UNEQUAL")
ELSE
  WriteString ("EQUAL")
END;
```

Try compiling the following C-program with different C compilers or at different optimization levels:

```
main()
{
int x,y;
x = 3;
y = (x = 2 * x) + (x = x + 1);
printf("x: %d, y: %d\n", x, y);
}
```

If you get different results, can you explain the differences ?
Are the compilers correct ?

=====