Figure 5.4
Example slices in FORTRAN 90

MAT (1:3, 2)

MAT (2:3, 1:3)

CUBE (2, 1:3, 1:4)

CUBE (1:3, 1:3, 2:3)
COBOL Program

MOVE COPY-RECORD 01 IN-RECORD 01.

MOVE RECORD-RECORD 01.

MOVE RECORD-RECORD 01.

MOVE RECORD-RECORD 01.

MOVE RECORD-RECORD 01.

The file contains a COBOL program that reads data from an input record and stores it in a corresponding output record. The program includes declarations for various fields, such as address, position, type, name, and others, which are used to store the data. The program also includes record declarations and field definitions that are part of the COBOL language.
```pascal
type
  shape = (circle, triangle, rectangle);

var
  figure : shape;

begin
  case figure of
    circle : writeln ("It's a circle; the diameter is ");
    triangle : writeln ("It's a triangle; the sides are ");
    rectangle : writeln ("It's a rectangle; the angles are ");
  end;
end.
```

This is an example of how to use a variant record in Pascal.
Implementation of union types:

```pascal
type NODE (TAG : BOOLEAN) is
record
  case TAG is
  when true => COUNT : INTEGER;
  when false => SUM : FLOAT;
end case;
end record;
```

**Figure 5.10**
A compile-time descriptor for a discriminated union

```
<table>
<thead>
<tr>
<th>TAG</th>
<th>Discriminated union</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>true</th>
<th>Case table</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>COUNT</td>
</tr>
<tr>
<td>type</td>
<td>INTEGER</td>
</tr>
</tbody>
</table>

| false | |
| name | SUM |
| type | FLOAT |
```

**Pascal's set type:**

```pascal
type colors = (red, blue, green, yellow, orange, white, black);
colorset = set of colors;
var set1, set2 : colorset;

Constant values can be assigned to the set variables set1 and set2, as in

set1 := [red, blue, yellow, white];
set2 := [black, blue];
```
Figure 5.13

Algorithm

actions of the marking
an example of the

end if

end if

mark(per[1].link)
mark(per[2].link)
set ptr.leg
if ptr.leg is not marked then
else
null

procedure mark(ptr)

mark(x)
for every pointer l do

(a) with linkages

(b) without linkages

Heap

Dynamic

Heap

Heap

Dynamic

Heap