/* an ANSI C function declaration gives the function name and argument types */
int min(int arrInt[], int size); /* min() is defined in another file to be linked with this one */

int main()
{
    const int NUM_SCORES = 5;
    int scores[] = { 70, 85, 75, 55, 90 };
    int worst;
    /* the compiler uses the function declaration to determine whether the invocation is */
    /* syntactically correct */
    worst = min(scores, NUM_SCORES);
}

/* a declaration which is not a definition in C */
extern int arrSize;
10. Assume you are given the following program:

```fortran
PROGRAM main;

VAR x: INTEGER;

PROCEDURE sub1;
BEGIN ( sub1 )
    WRITELN('x = ', x)
END; ( sub1 )

PROCEDURE sub2;
VAR x: INTEGER;
BEGIN ( sub2 )
    x := 2;
    sub1
END; ( sub2 )

BEGIN ( main )
    x := 1;
    sub2
END. ( main )
```

a) What value of x is printed by the WRITELN in sub1 if static scoping is used?

\[ 1 \]

b) What value of x is printed by the WRITELN in sub1 if dynamic scoping is used?

\[ 2 \]
/ example of function overloading in C++ */
/* each of the following functions are declared here and are defined elsewhere */
void print(Date);
void print(int arrint[], int size);
void print(Employee);

int main()
{
    Date birthday;
    const int NUM_SCORES = 30;
    int scores[NUM_SCORES];
    Graph routes;
    /* ... other operations ... */
    /* the compiler determines which function to call by the argument type(s) */
    print(birthday);    /* print(Date) invoked */
    print(scores, NUM_SCORES);    /* print(int[], int) invoked */
    print(routes);    /* error: print(Graph) not declared */
}

<table>
<thead>
<tr>
<th>source code (compile time)</th>
<th>storage (execution time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>location (&quot;l-value&quot;)</td>
</tr>
<tr>
<td>binding</td>
<td>value (&quot;r-value&quot;)</td>
</tr>
</tbody>
</table>

Figure 0.5: The components of a variable
\begin{verbatim}
\{ a record type and three instances in Pascal \}
type
  Point =
    record
      x: integer;
      y: integer
    end
var
  pt1, pt2, pt3: Point;
\end{verbatim}

\begin{verbatim}
/* a struct type and instances in C */
struct Point /* data type "struct Point" */
{
  int x;
  int y;
} pt1, pt2, pt3; /* three instances */
\end{verbatim}
-- An Ada array type and two instances with different bounds
type VECTOR is array(INTEGER range < >) of FLOAT;
ARR1: VECTOR(-100 .. 100);
ARR2: VECTOR(0 .. 50);

-- an Ada function with an array parameter
function SUM(VEC: VECTOR) return FLOAT is
  TOTAL: FLOAT := 0.0;
begin
  for INDEX in VEC'FIRST .. VEC'LAST loop
    TOTAL := TOTAL + VEC(INDEX);
  end loop;
  return TOTAL;
end SUM;

{ a variant record in Pascal }
type
  ShapeType = (Circle, Rectangle, Spiral);
  Shape =
    record
      center: Point;  { Point defined in section 0.3.3 }
      case typeTag: ShapeType of
        Circle: (radius: integer);
        Rectangle: (height, width, tilt: integer);
        Spiral: (radius, spacing: integer)
      end
end

-- type safety for variant records in Ada
GRAPHIC: SHAPE;
GRAPHIC := (CIRCLE, (2, 2), 5);  -- a CIRCLE literal specification
GRAPHIC.TYPE := SPIRAL;         -- illegal, not a complete specification
GRAPHIC.HEIGHT := 5;            -- illegal, GRAPHIC is a CIRCLE
GRAPHIC.RADIUS := 3;            -- OK

/* an undiscriminated union in C */
union Data
{
  int in;
  char ch;
  double dbl;
} data1, data2;

{ a function parameter in Pascal }
function slope(function f(x: real): real; x1, x2: real): real;
begin
  if x1 = x2 then
    slope := 0
  else
    slope := (f(x2) - f(x1)) / (x2 - x1)
  end

{ passing a function object as an argument to a higher-order function }
m := slope(sin, 3.0, 5.0);