Overview. A symbol table is created for a compilation unit. A compilation unit is usually a single source file but may include any number of other files included into the source file by directives in the source file.

The symbol table is two maps.

1. The symbol table maps identifiers to a relative address.
2. The symbol table maps identifiers to their type.

Locality. Locality serves to distinguish duplicate identifiers. The usual technique is that a reference to an identifier that is defined in two or more separate locations is to assign the reference to the nearest definition.

For example, consider the following Subset Pascal program in figure 1.

```
 1.  program p00;
 2.      var i:integer;
 3.      procedure p
 4.          (b:boolean;c:char);
 5.          var r:real;
 6.      begin{p}
 7.      end{p};
 8.      function f
 9.          (i:integer;r:real):real;
10.          var b:boolean;
11.      begin{f}
12.          f:=1.602e-19
13.      end{f};
14.  begin{p00}
15.  end{p00}.
```

Figure 1. program **p01**.

Variable *i* is defined in two locations, on line 2 and again as a parameter on line 9. The nearest declaration of variable *i* is the one selected if it appears in a statement in program *p00*. Inside function f, variable i as declared on line 9 is selected. Everywhere else in program *p00*, the declaration of variable *i* on line 2 is selected.

The search for an identifier begins with the current locality and proceeds through all the enclosing localities. Each locality defines a namespace. All identifiers in a namespace are unique. In the example above there are four localities. There is the base locality of predefined identifiers that includes the names of types, *boolean*, *char*, *integer*, *real*, and others. The program name, p00, is also stored in this locality.

There is a locality defined by the program that includes identifiers, *i*, a variable (line 2) *p*, a procedure (line 3), and *f*, a function (line 8).

There is a locality for procedure *p* that includes parameters *b* and *c* and variable *r*.

There is a locality for function *f* that includes parameters *i* and *r* and variable *b*.

A stack can be used to facilitate the process of searching localities as shown in figure 2. The current locality is on top of the stack. Enclosing localities appear under the current locality.

Lexical Level	Namespace	Stack Index	Identifiers
2	*p*	2	*b, c, r*
1	*p00*	1	*i, p, f*
0	*predefined*	0	*boolean, char, integer, real, false, true, …*
Figure 2. 1 Stack of Localities at line 6			

Lexical Level	Namespace	Stack Index	Identifiers
2	*f*	2	*i, r, b*
1	*p00*	1	*i, p, f*
0	*predefined*	0	*boolean, char, integer, real, false, true, …*
Figure 2.2 Stack of Localities at line 12			

Predefined Types:

Boolean		
typekind	*size*	*alignment*
Typekind	**int**	**int**
tk_boolean	**8**	**8**

Character_8		
typekind	*size*	*alignment*
Typekind	**int**	**int**
tk_character	**8**	**8**

Figure 3.1 Typical Scalar Types, Boolean and Character

Integer_32		
typekind	*size*	*alignment*
Typekind	**int**	**int**
tk_integer	**32**	**32**

Real_32		
typekind	*size*	*alignment*
Typekind	**int**	**int**
tk_real	**32**	**32**

Figure 3.2 Typical Scalar Types, Integer and Real

Scalar and Subrange Types:

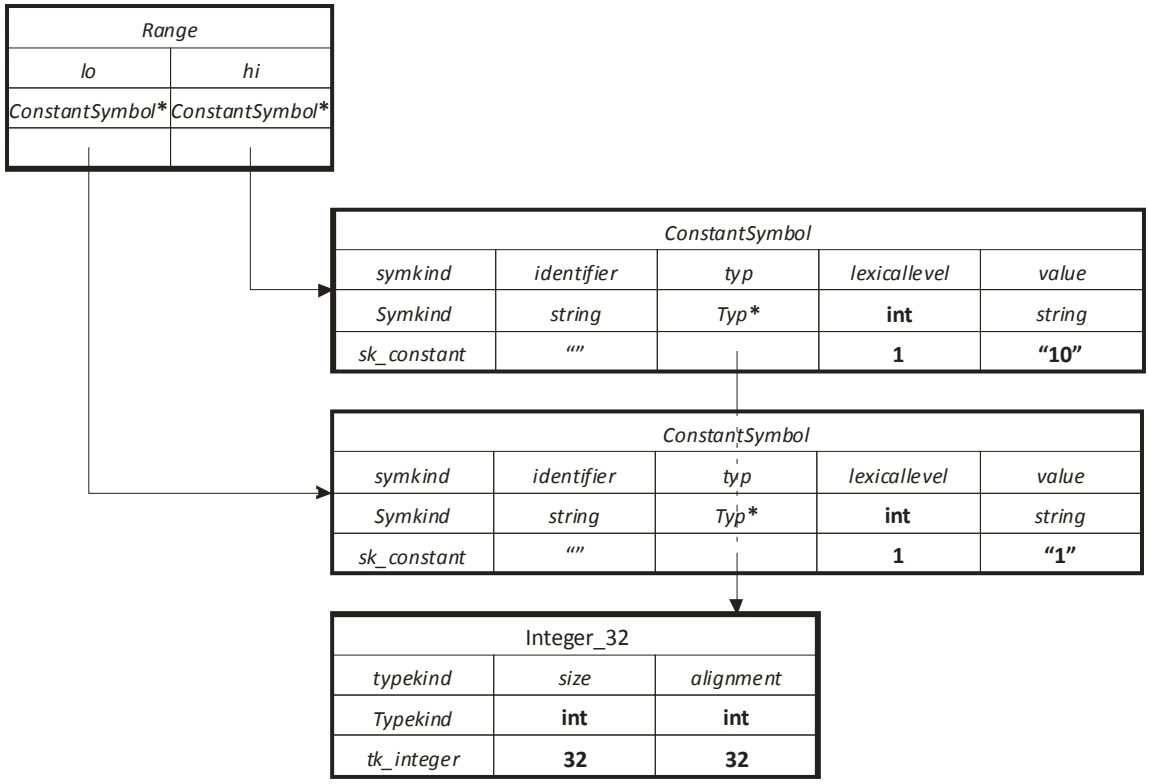

Figure 3.3 Range Types

var *a*:1..10;

Structured Types:

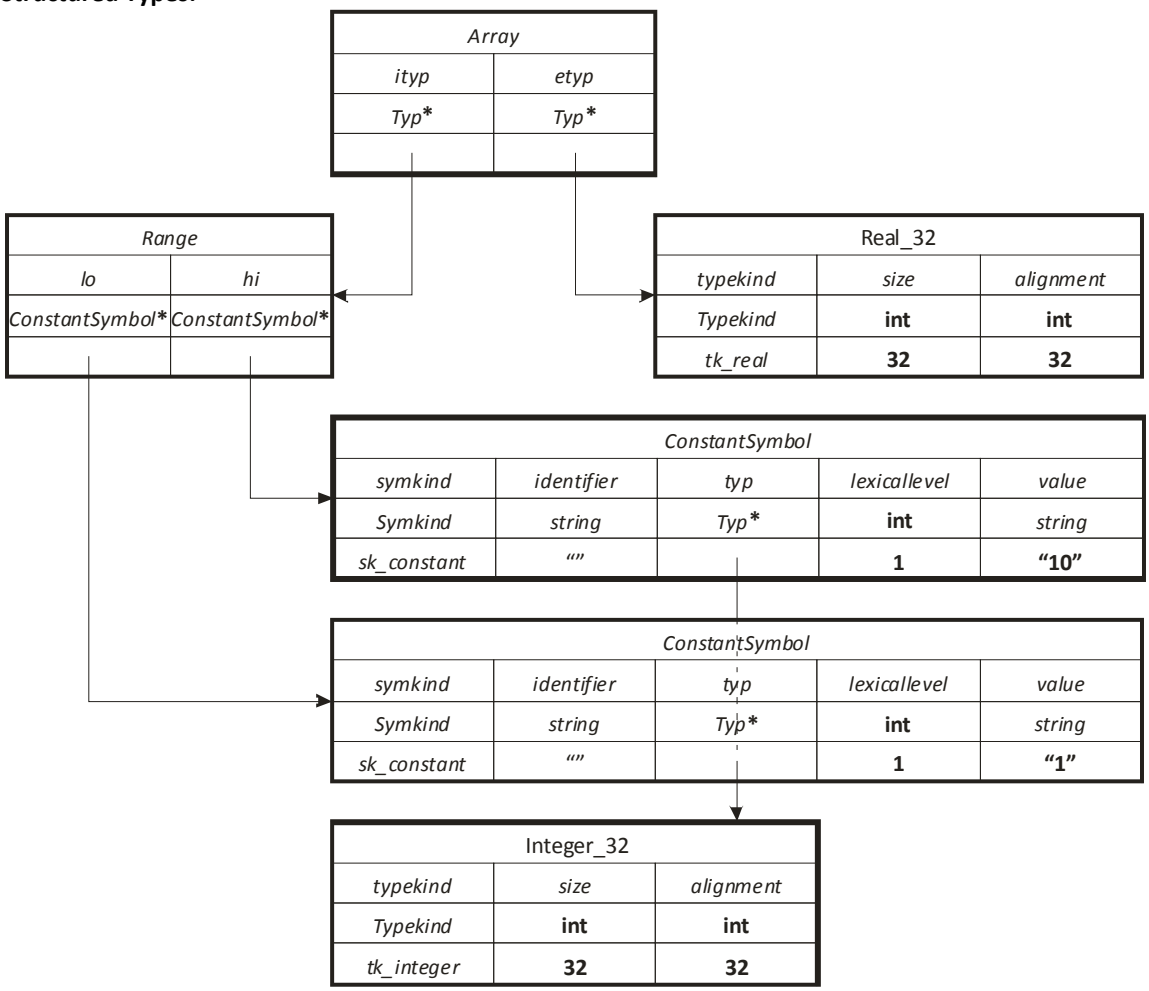

Figure 3.4 Array Types

var *a*:**array[1..10] of** *real*;

Variable Symbols: