CSL302: Compiler Design

Symbol Table

Vishwesh Jatala

Assistant Professor

Department of CSE

Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

Acknowledgement

- References for today's slides
  - *National Taiwan University:*
    - *https://www.csie.ntu.edu.tw*
  - *Lecture notes of Prof. Amey Karkare (IIT Kanpur) and Late Prof. Sanjeev K Aggarwal (IIT Kanpur)*
  - *Suggested textbook for the course*

Symbol Table

- Symbol table: A data structure used by a compiler to keep track of semantics of names.
  - Data type
  - When is used: scope.
    - The effective context where a name is valid.
- Operations:
  - **Search**: whether a name has been used.
  - **Insert**: add a name.
  - **Delete**: remove a name when its scope is closed.

Symbol Table: Entries

- Possible entries in a symbol table:
  - Name: a string.
  - Attribute:
    - Variable
    - Procedure
    - Constant
  - Data type.
  - Storage allocation, size, . . .
  - Scope information: where and when it can be used.

Symbol Table: Implementations

- Unordered list:
    - for a very small set of variables;
    - coding is easy, but performance is bad for large number of variables.
- Ordered linear list:
    - use binary search;
    - insertion and deletion are expensive;

Symbol Table: Implementations

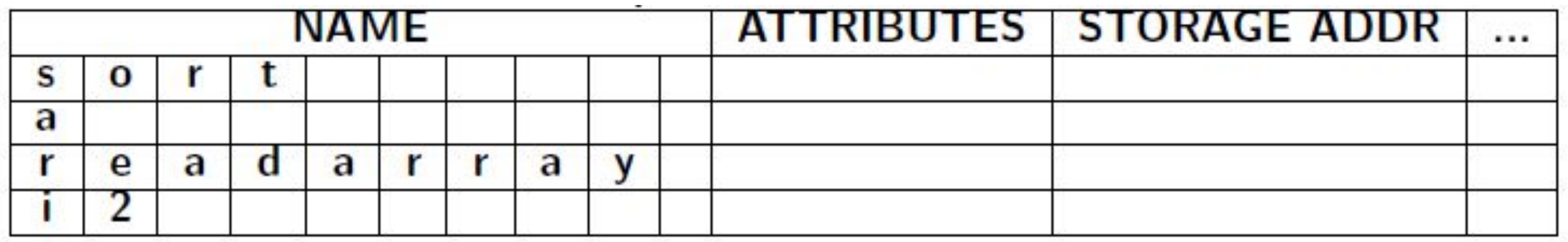
- Binary search tree:
    - O(log n) time per operation (search, insert or delete) for n variables;

Symbol Table: Implementations

- Hash table:
  - most commonly used;
  - very efficient provided the memory space is adequately larger than the number of variables;
  - performance maybe bad if unlucky or the table is saturated;

Symbol Table: Representations

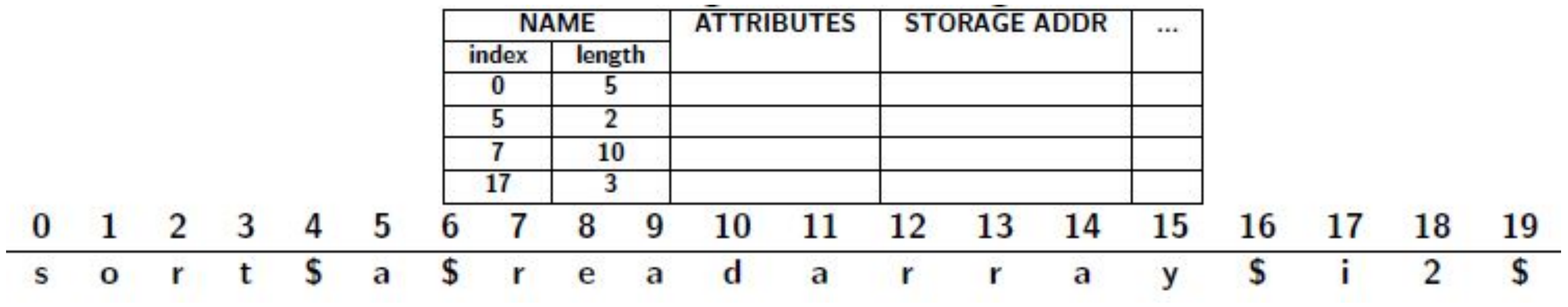
- Fixed-length name: allocate a fixed space for each name allocated.
    - Too little: names must be short.
    - Too much: waste a lot of spaces



NAME										ATTRIBUTES	STORAGE ADDR	...
s	o	r	t									
a												
r	e	a	d	a	r	r	a	y				
i	2											

Symbol Table: Representations

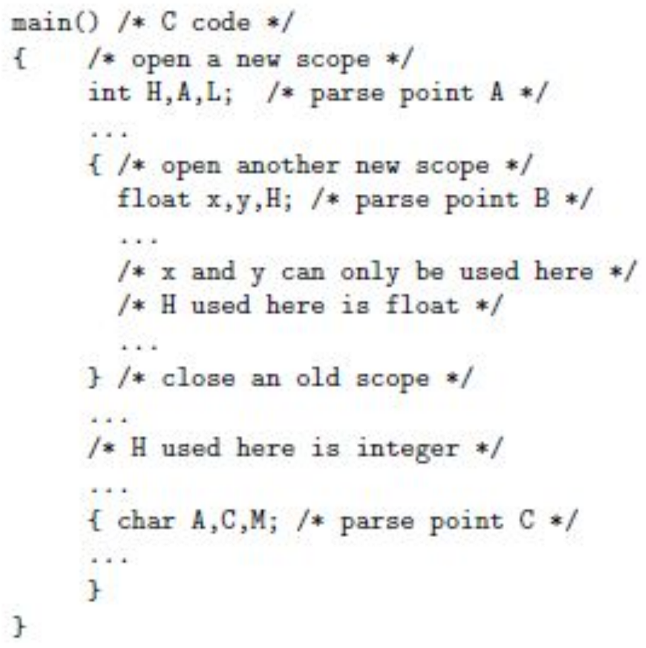
- Variable-length name:
  - A string of space is used to store all names.
  - For each name, store the length and starting index of each name.



NAME		ATTRIBUTES	STORAGE ADDR	...
index	length			
0	5			
5	2			
7	10			
17	3			

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
s	o	r	t	$	a	$	r	e	a	d	a	r	r	a	y	$	i	2	$

Handling Block Structures



```
main() /* C code */
{    /* open a new scope */
     int H,A,L;  /* parse point A */
     ...
     { /* open another new scope */
       float x,y,H; /* parse point B */
       ...
       /* x and y can only be used here */
       /* H used here is float */
       ...
     } /* close an old scope */
     ...
     /* H used here is integer */
     ...
     { char A,C,M; /* parse point C */
     ...
     }
}
```

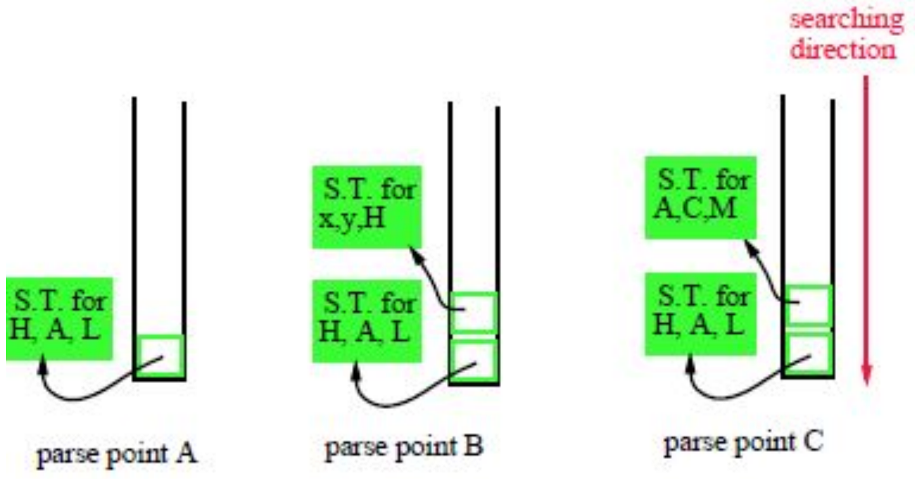
- Nested blocks mean nested scopes.
- Two major ways for implementation:
  - Approach 1: multiple symbol tables in one stack.
  - Approach 2: one symbol table with chaining.

Multiple Symbol Tables in One Stack

- An individual symbol table for each scope.
    - Use a stack to maintain the current scope.
    - Search top of stack first.
    - If not found, search the next one in the stack.
    - Use the first one matched.

Handling Block Structures

```
main() /* C code */
{    /* open a new scope */
     int H,A,L;  /* parse point A */
     ...
     { /* open another new scope */
       float x,y,H; /* parse point B */
       ...
       /* x and y can only be used here */
       /* H used here is float */
       ...
     } /* close an old scope */
     ...
     /* H used here is integer */
     ...
     { char A,C,M; /* parse point C */
     ...
     }
}
```



Pros and Cons

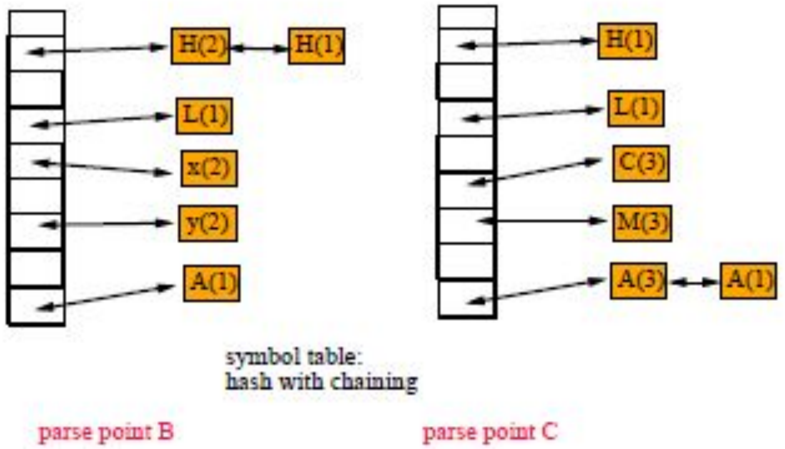
- Advantage:
  - Easy to close a scope.
- Disadvantage:
  - Difficulties encountered when a new scope is opened
  - Searching overhead

One Symbol Table With Chaining

- A single global table marked with the scope information.
    - Each scope is given a unique scope number.
    - Incorporate the scope number into the symbol table.
- Hash table with chaining.

Handling Block Structures

```
main() /* C code */
{     /* open a new scope */
      int H,A,L;  /* parse point A */
      ...
      { /* open another new scope */
        float x,y,H; /* parse point B */
        ...
        /* x and y can only be used here */
        /* H used here is float */
        ...
      } /* close an old scope */
      ...
      /* H used here is integer */
      ...
      { char A,C,M; /* parse point C */
      ...
      }
}
```



Pros and Cons

- Advantage:
  - Does not waste space.
  - Little overhead in opening a scope.
  - Searching is constant time
- Disadvantage:
  - It is difficult to close a scope.

Questions?