UNIT-3

1 question is asked for 5 marks from this unit

SYMBOL TABLE DESIGN AND RUNTIME STORAGE MANAGEMENT

upto functions

⊛. Symbol Table Design: [Imp]

Symbol Table is an important data structure created and maintained by the compiler in order to keep track of semantics of variable. It is built in lexical and syntax analysis phase. It is used by various phases of compiler as follows:

Lexical Analysis: Creates new table entries in the table, example like entries about token.

Syntax Analysis: Adds information regarding attribute type, scope, dimension, line of refrence, use etc. in the table.

Semantic Analysis: Uses available information in the table to check for semantics.

Intermediate Code generation: Helps in adding temporary variable information.

Code Optimization: Uses information present in symbol table for machine dependent optimization.

Target Code generation: Generates code by using address information of identifier present in the table.

Functions of a symbol table:

→To store the names of all entities in a structured form at one place.
→To verify if a variable has been declared or not.
→To determine the scope of a name (scope resolution).
→To access information associated with a given name.
→To add new information with a given name.
→To delete a name or group of names from the table.

⊛. Information Used by Compiler from Symbol Table:

Name:
→ Name of identifier.
→ May be stored directly or as a pointer to another character string in an associated string table. names can be arbitrarily long.

Type:
→ Type of identifier: variable, label, procedure name etc.
→ For variable, it's type: basic types, derived types etc.

Location:
→ Offset within the program where the current definition is valid.

Other attributes: array limits, fields of records, ~~pattern~~ parameters, return values etc.

Scope:
→ Region of the program where the current definition is valid.

⊛ Basic Operations on a symbol table:

allocate → to allocate a new empty symbol table.

free → to remove all entries and free the storage for symbol table.

insert → to insert a name in a symbol table and return a pointer to it's entry.

lookup → to search for a name and return a pointer to it's entry.

set_attribute → to associate an attribute with a given entry.

get_attribute → to get an attribute associated with a given entry.

⊛. Storing Names in Symbol Table:

There are two types of name representation:

1. Fixed Length Name: A fixed space for each name is allocated in symbol table. In this type of storage if name is too small then there is wastage of space.

2. Variable Length Name: The amount of space required by string is used to store the names. The name can be stored with the name of starting index and length of each name.

⊛. Data Structures for Symbol Table:

Following are the commonly used data structures for implementing symbol table:

1). List Data Structure: In this method, an array is used to store names and associated information. A pointer "available" is maintained at end of all stored records and new names are added in the order as they arrive.

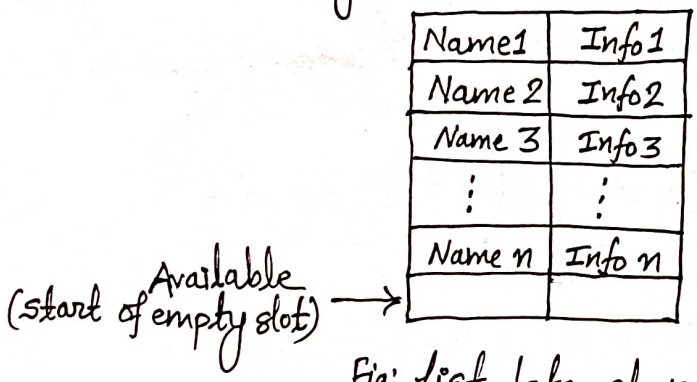

Fig: list data structure for symbol table.

2) Self Organizing List Data Structure: This implementation is using linked list. A link field is added to each record. A pointer "First" is maintained to point of first record of symbol table. Searching of names is done in order pointed by link of link field.

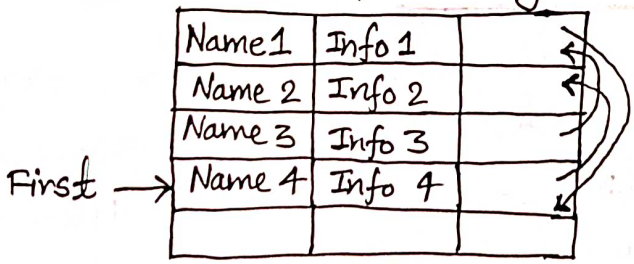

Fig: Self organizing list data structure for symbol table.

3) Binary Trees: When the organization of symbol table is by means of binary tree, the node structure will be as follows:

Left child	Symbol	Information	Right child

The left child field stores the address of previous symbol and right child field stores the address of next symbol. The symbol field is used to store the name of symbol, and information field is used to give information about symbol.

4) Hash Tables: In hashing scheme two tables are maintained: A Hash Table and Symbol Table. The hash table consist of k entries from 0,1, to k-1. These entries are basically pointers to symbol table pointing to the names of symbol table. To determine whether the 'Name' is in symbol table, we use a hash function 'h' such that h(name) will result any integer between 0 to k-1. We can search any name by

position = h(name)

Using this position we can obtain the exact location of name in symbol table.

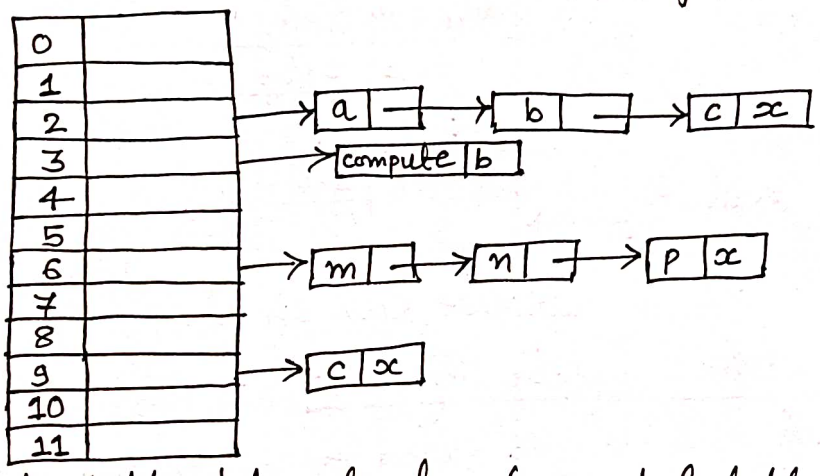

Fig: Hash table data structure for symbol table.

⊛. Run-time Storage Management:

Activation record: Activation record is a block of memory used for managing information needed by a single execution of a procedure. The following three-address statements are associated with the run-time allocation and de-allocation of activation records:

1. Call
2. Return
3. Halt
4. Action, a placeholder for other statements.

Activities performed by caller and callee during procedure call and return:

On caller's side:

→ Save registers for later use.
→ Arrange for the procedure arguments to be found by the procedure.
→ Call the procedure.
→ Copy the return value.
→ Throw away procedure arguments.
→ restore saved registers
→ continue

On the callee's side:

→ Control is transfered from the caller to the starting address.
→ Save registers for later use.
→ allocate storage for automatic local variables.
→ allocate storage for temporaries.
→ put return value in appropriate place.
→ Throw away local variables and temporaries.
→ restore saved registers
→ work

Different storage allocation strategies are:-

compile time

1) Static Storage Allocation: In static allocation, if memory is created at compile time, memory will be created in the static area and only once. It does not support dynamic data structure i.e, memory is created at compile time and de-allocated after program completion. The drawback with static storage allocation is recursion is not supported and size of data should be known at compile time. It's advantage is, it is faster ~~that~~ than stack storage allocation.

run-time

2) Stack Storage Allocation: Stack allocation is based on the idea of control stack.

↳ A stack is Last In First Out storage device where new storage is allocated and de-allocated at only one end, called the top of the stack.

↳ Storage is organized as a stack and activation records are pushed and popped as activations begin and end, respectively.

↳ Storage for the locals in each call of procedure is contained in the activation record for that call.

↳ The values of local are detected when the activation ends.

↳ At ~~run~~ run time, an activation record can be allocated and de-allocated by incrementing and decrementing top of the stack respectively.

Advantages:

→ It supports recursion as memory is always allocated on block entry.

→ It allows creating data structure dynamically.

Disadvantage:

→ Memory addressing can be done using pointers and index registers. Hence, this type of allocation is slower than static allocation.

run-time

3) Heap Storage Allocation: The de-allocation of activation records need not occur in a last-in first-out fashion, so storage cannot be organized as a stack. Heap allocation parcels out pieces of contiguous storage, as needed for activation records or other objects. Pieces may be de-allocated in any order. So over time the heap consist of alternate areas that are free and in use. Heap is an alternate for stack.