

# Symbol Table

## C Structure for type\_info

```
typedef enum type_kind TY_KIND;
enum type_kind {Integer, Char, Boolean, String, SubRange, Array,
                Record, UndefinedType};
typedef struct type_info TYPE;
struct type_info
{
    int ty_size;
    TY_KIND ty_kind;
    TYPE *ty_next;
    union
    {
        struct
        {
            TYPE *RangeType;
            int min;
            int max;
        } ty_subrange;
        struct
        {
            TYPE *ElementType;
            TYPE *IndexType;
        } ty_array;
        struct
        {
            ID *FirstField;
        } ty_record;
    } ty_form;
}; /* type_info */
```

CS 5300 - SJAllan

2

# Function typecreate

```
TYPE *typecreate (int size, TY_KIND kind, ID *id_list, TYPE *type)
{
    TYPE *t;
    t = (TYPE *)malloc(sizeof(TYPE));
    t->ty_size = size;
    t->ty_kind = kind;
    t->ty_next = NULL;
    switch (kind)
    {
        case SubRange:
            t->ty_form.ty_subrange.RangeType = type;
            break;
        case Array:
            t->ty_form.ty_array.ElementType = type;
            break;
        case Record:
            t->ty_form.ty_record.FirstField = id_list;
            break;
        default:
            break;
    } /* switch */
    return(t);
} /* typecreate */
```

CS 5300 - SJAllan

3

# TYPE Global Variables

```
TYPE *int_type, *bool_type, *char_type,
      *str_type, *undef_type;
```

CS 5300 - SJAllan

4

## Function typeinit

```
void typeinit (void)
{ int_type = typecreate(INTSIZE, Integer, NULL, NULL);
  bool_type = typecreate(BOOLSIZE, Boolean, NULL, NULL);
  char_type = typecreate(CHARSIZE, Char, NULL, NULL);
  str_type = typecreate(POINTSIZ, String, NULL, NULL);
  undef_type = typecreate(NOSIZE, UndefinedType, NULL, NULL);
} /* typeinit */
```

## Size Constants

```
#define INTSIZE 4 /* integer data */
#define CHARSIZE 1 /* character data */
#define BOOLSIZE 1 /* boolean data */
#define POINTSIZE 4 /* pointer data */
#define NOSIZE 0 /* unknown data */
```

## Symbol Table Organization

- Symbol table must accommodate scoping rules
  - Identifier names already in use in an enclosing scope must be allowed to be reused without conflict
  - When searching for an identifier, the most recent definition must be found
  - Must be able to accommodate the removal of all identifiers that are associated with a given scope when the scope is closed

## Symbol Table Organization

- A symbol table for a scope is represented as a binary search tree
- A stack of symbol tables is maintained
  - Each entry in the stack points to the root of the tree for each currently active scope
- We follow the following conventions:
  - Scope 0 – predefined identifiers in the language
  - Scope 1 – global (main) level
  - Scope 2 – inside procedures and functions
- The variable `currscope`, initialized to zero, tells us which scope we are currently in

# Example

```
1-> program X;
    type a = array [1..10] of boolean;
    var i, j : integer;
2->   procedure p1 (a, b : integer);
        var i, k : boolean;
        begin
            .....
1->   end; { p1 }
2->   procedure p2 (x, y : boolean);
        var j, b : boolean;
3->     function f (v1, v2 : char) : integer;
        var b : char;
        begin
            .....
2->     end; { f }
        begin
            .....
1->   end; { p2 }
        begin
            .....
0-> end. { X }
```

CS 5300 - SJAllan

9

# Symbol Table Stack

```
#define SCOPEDEPTH 3
ID *scope [SCOPEDEPTH];
int currscope;
```

CS 5300 - SJAllan

10

# Symbol Table Entry

```
typedef enum identifier_kind ID_KIND;
enum identifier_kind {Constant, Type, Variable, RParameter, VParameter,
                     Field, Procedure, Function};

typedef struct id_info ID;

struct id_info
{  char *id_name;
   int id_addr;
   int id_level;
   TYPE *id_type;
   ID *id_left;
   ID *id_right;
   ID_KIND id_kind;
   ID *id_next;
   int id_value;
}; /* id_info */
```

CS 5300 - SJAllan

11

# Function newid

```
ID *newid (char *name)
{  ID *new_id;

   new_id = (ID *)malloc(sizeof(ID));
   new_id->id_name = name;
   new_id->id_addr = 0;
   new_id->id_level = currscope;
   new_id->id_type = undef_type;
   new_id->id_left = NULL;
   new_id->id_right = NULL;
   new_id->id_kind = Variable;
   new_id->id_next = NULL;
   new_id->id_value = 0;
   return(new_id);
} /* newid */
```

CS 5300 - SJAllan

12

## Function search

```
ID *search (char *name, ID *table)
{  int temp;

    if (table == NULL)
        return(NULL);
    temp = strcmp(name, table->id_name);
    if (temp == 0)
        return(table);
    else if (temp < 0)
        return(search(name, table->id_left));
    else
        return(search(name, table->id_right));
} /* search */
```

CS 5300 - SJAllan

13

## Type Initialization

```
entertype(entertype(newid(strdup("INTEGER"))), int_type, Type);
entertype(entertype(newid(strdup("boolean"))), bool_type, Type);
```

CS 5300 - SJAllan

14

## Name Initialization

```
st_temp = newid(strdup("TRUE"));  
st_temp->id_value = 1;  
entertype(entertype(st_temp), bool_type, Constant);
```