Compilers – lab5

Outline

- Tree Traversal
- Symbol Table
- Abstract Data Types
- Report

Tree Traversal

- Syntax tree data structure (C implementation)

Tree Traversal

- Syntax tree data structure (n- ary tree)

```
typedef struct node {
    int val;
    struct node *t1;
    struct node *t2;
    struct node *t3;
    struct node *t4;
} node;

node *node_init(int val, node *t1, node *t2, node *t3, node *t4);
```

Tree Traversal

- Syntax tree data structure (n- ary tree)

```
void node_traverse(node *self){
    printf("%d\n", self->val);
    if (self->t1 != NULL) node_traverse(self->t1);
    if (self->t2 != NULL) node_traverse(self->t2);
    if (self->t3 != NULL) node_traverse(self->t3);
    if (self->t4 != NULL) node_traverse(self->t4);
}
```

Tree Traversal

- Syntax tree data structure (child- sibling tree)

```c
typedef struct node {
    int val;
    struct node *child;
    struct node *sibling;
} node;

node *node_init(int val, int num_children, ...);
```

Tree Traversal

- Syntax tree data structure (child- sibling tree)

```
void node_traverse(node *self){
    node *tmp;
    printf("%d\n", self->val);
    tmp = self->child;
    while(tmp != NULL){
        node_traverse(tmp);
        tmp = tmp->sibling;
    }
}
```

Tree Traversal

- Syntax tree data structure (variadic children)

```c
typedef struct node {
    int val;
    int num_children;
    struct node **children;
} node;

node *node_init(int val, int num_children, ...);
```

Tree Traversal

- Syntax tree data structure (variadic children)

```
void node_traverse(node *self){
    printf("%d\n", self->val);
    for(int i = 0; i < self->num_children; i++){
        node_traverse(self->children[i]);
    }
}
```

Tree Traversal

- Visitor pattern

```
int visit_Program(ast *program);
int visit_ExtDefList(ast *extDefList);
int visit_ExtDef(ast *extDef);
int visit_ExtDecList(ast *extDecList);
// ...
```

Tree Traversal

- Visitor pattern

```
int visit_ExtDecList(ast *extDecList){
    /* for rule:
       ExtDecList : VarDec
                  | VarDec COMMA ExtDecList
    */

    // do something
    if(extDecList->num_children == 1){
        visit_VarDec(extDecList->children[0]);
    }
    else if(extDecList->num_children == 3){
        visit_VarDec(extDecList->children[0]);
        visit_ExtDecList(extDecList->children[2]);
    }
    // do something
}
```

Symbol Table

- A *symbol table* maps a name to its associated information.
  - **name**:

    variable name, function name, user-defined type name, …
  - **information**:

    types, array dimension, struct members, initial values, …

Symbol Table

- **Symbol table operations**
  - **lookup**:

    check for variable existence, type definition, …
  - **insert**:

    meet function/variable declaration, …
  - ***delete***:

    current scope popped, delete all identifiers inside

Symbol Table

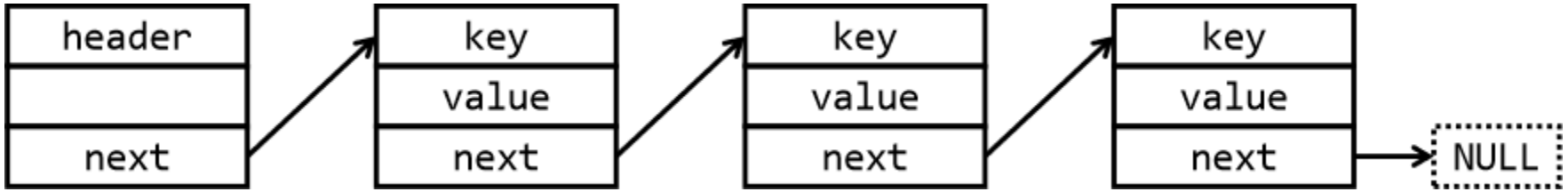
- Implement symbol table in terms of:
  - stored information (suggest to store only types)
  - implementation: linked list, hash table, binary tree, …

Abstract Data Types

- Typically key-value pairs (no duplicate keys)
  - linked list
  - binary tree
  - hash table

Abstract Data Types

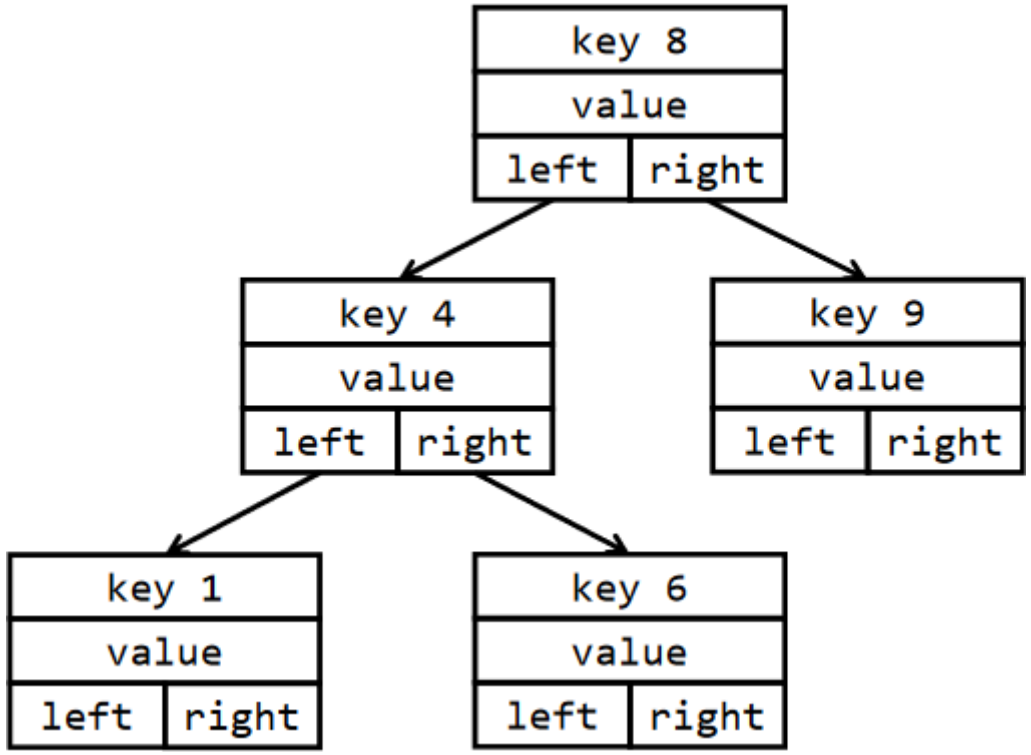
- **Linked list**



  - lookup: O(n) in worst case
  - insert: O(1) at head, O(n) at tail
  - delete: O(n) in worst case

Abstract Data Types

- **Binary tree**
  - ideally, operations are O(log n)
  - O(n) in worst case
  - balance strategies:
    - + AVL tree
    - + red-black tree



Abstract Data Types

- **Hash table**
  - compress key to index (hash function)
  - hash conflict: open/close addressing
  - most operations can be done in O(1)
  - drawback: space consumption

ADT Exercises

- In `lab5` directory
  - there are three .c files
  - Read the code in symtab_ll.c
  - Make the target for symtab_ll.c

    command: make lltest
  - Optional: write your code in symtab_bt.c or symtab_ht.c and make the corresponding target **(no bonus point)**

    command: make bttest

    command: make httest

Report

1.Academic Integrity: Plagiarism or any form of cheating is strictly prohibited. Your work should be original, and any external sources should be appropriately cited.
2.Programming Assignments: Feel free to ask questions and seek assistance from the teaching assistant if needed.
3.Report:
- •Pdf type
- •Naming like **name_studentID.pdf**
- •Include any relevant diagrams, charts, or screenshots to enhance your explanations.
- •Make sure your report is well-structured, with appropriate headings and subheadings.

4.Submission Guidelines:
- •Include your **name, student ID, class number** and **container number** in the report's header. For Docker on Windows systems, you can view the container numbers in the containers of the Docker Desktop.
- •Commit the compressed package of the lab5 folder（**lab5.zip**）, which should include the code, the compiled results, and your PDF report.

5.Deadline:
December 3, 2023, 23:59
6.Submission Platform:
Teaching cloud platform