

Data Structures

Declaring a struct:

```
typedef struct {
    int x;
    int y;
} point;
```

Declaring a variable and accessing members:

```
point first;
first.x = 1;
first.y = 4;
printf ("%d, %d \n",
first.x, first.y);
```

Point is name of struct.

Omega

	lower (Ω)	upper (O)
insertion into a hash table with separate chaining	1	1
insertion into a trie	1	1
insertion into a sorted linked list	1	n
deletion from a sorted linked list	1	n
deletion from an unsorted linked list	1	n

Common Structs

Hashtable:

```
typedef struct _node
{
    char word[50]; //
    50-char word
    struct _node
    *next;
}
node;
```

Tree:

```
typedef struct _tree3 {
    bool valid; //
    exists or not
    struct _tree3
    *child1;
    struct _tree3
    *child2;
    struct _tree3
    *child3;
}
tree3;
```

Trie:

```
typedef struct _btrie {
    bool valid;
    struct _btrie
    *child ren[2];
}
btrie;
```

Stacks

Pop:

```
int pop(void)
{
    if (stack.size == 0)
        return -1;
    return stack.num ber -
s[- -st ack.size];
```

Stacks (cont)

>}

Push:

```
bool push(int n)
{
    if (stack.size == CAPACITY || n < 0)
        return false;
    stack.numbers[stack.size++] = n;
    return true;
}
```

Pointers

Declaration and initialization:

```
int a = 14;
int b = 15;
int * iPtr;
iPtr = &a;
int * anotherPtr = &b;
```

Accessing pointers and values:

```
// assign an address to another
pointer
ano therPtr = iPtr;
// change the value stored in
the memory
// location being pointed to
*iPtr = 3;
// print the address held be a
pointer
pri ntf ("%x \n", iPtr);
// print the value being pointed
to
pri ntf ("%d \n",
*iPtr);
```

&b = "address of" operator

*iPtr = dereference operator

iPtr -> a = 14; //shortcut

Definitions

Valgrind: used for detecting memory leaks from forgetting to `fclose()` and `free()`

- syntax: `valgrind -v --leak-check=full <executable file>`

Bitwise Operators – see table to the right.

Find if a number is odd: `if (num & 1)`

`print("Odd");`

Hashtable - has 2 main parts: (1) a hash function, and (2) an array the hash function maps to. Often times, each index of the array will be a linked list to store the values that are hashed to a specific index. Struct of a hashtable node is below at left:

Tree - a data structure made up of nodes that have the following 2 rules: (1) A tree node can point at its children or at NULL, and (2) A tree node may not point at any other node other than those listed in (1), including itself. Struct of a 3-child tree is above right. In the diagram, black (top) is the root node and grey (point to NULL) are leave nodes. A binary tree is a special kind of tree that has 2 children left and right.

Trie – Just like tree but can have arbitrary number of children. Below are examples of binary trie and 6-child trie.

File Input / Output

Declaring a FILE pointer:

```
FILE * inputFile;
FILE * outputFile;
```

Opening a file:

```
inp utFile = fopen( " -
fil e1.t xt ", " r");
out putFile = fopen( " -
fil e2.t xt ", " w");
```

Input / Output:

```
fsc anf (in put File,
" %d", &x);
fpr int f(o utp utFile,
"%f \n", 3.14);
Closing a file:
fcl ose (in put File);
fcl ose (ou tpu tFile);
```

"r" for read

"w" for write

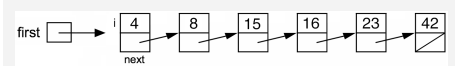
"a" for append

Operators

increment, decrement	++, --
multiply, divide, modulus	*, /, %
add, subtract	+, -
relational comparisons	>, >=, <, <=
equality comparisons	==, !=
and	&&
or	
assignment	=, +=, -=, *=, /=, %=

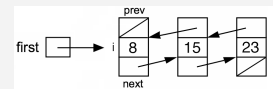
Grouped by precedence.

Linked Lists



Linked list is sorted with NULL pointer after 42.

Doubly Linked List



```
typedef struct node
```

```
{
    struct node* prev;
    unsigned int i;
    struct node* next;
}
node;
```