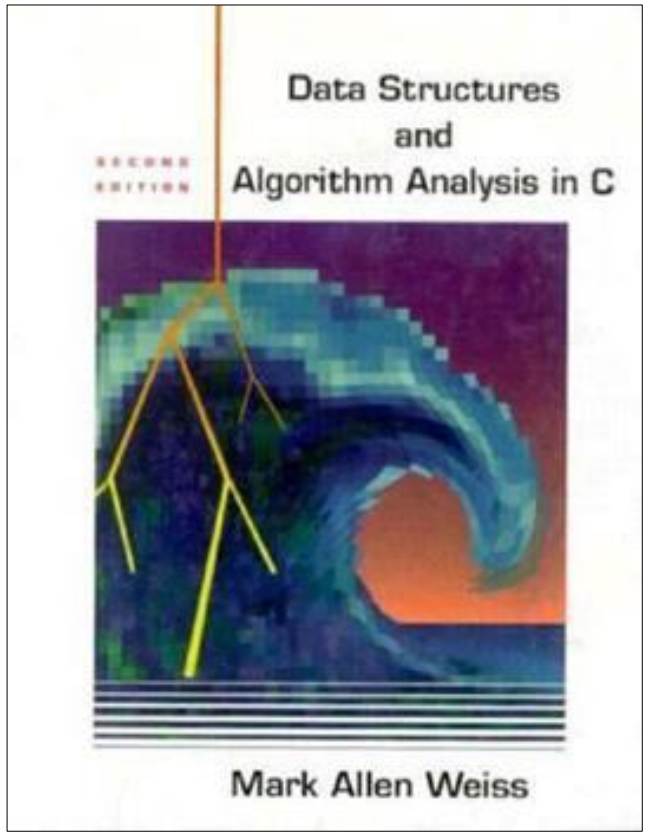
# Pointer-based data structures

E.g. linked lists, trees, graphs

No built-in data structures in C!

Nodes in these types of data structures use **structs**

**struct pointers** connect nodes together



# Exercise: Linked list in Java

What is a linked list?

What are some uses of linked lists? (e.g. when are they useful?)

How do you implement a linked list in Java?

# Example: Linked list C vs Java

```c
struct node {
  int val;
  struct node* next;
};

int main() {
  struct node n1 = {1, NULL};
  struct node n2 = {2, NULL};
  struct node n3 = {3, NULL};

  struct node* list;
  list = &n1;
  n1.next = &n2;
  n2.next = &n3;

  for (struct node* n = list; n != NULL; n = n->next) {
    printf("Val: %d\n", n->val);
  }
}
```

```java
class LinkedList {

  public static void main(String[] args) {
    Node n1 = new Node(1);
    Node n2 = new Node(2);
    Node n3 = new Node(3);

    Node list = null;
    list = n1;
    n1.next = n2;
    n2.next = n3;

    for (Node n = list; n != null; n = n.next) {
      System.out.println("Val: " + n.val);
    }
  }
}
```

# Visualize the linked list from this code

```
struct node {
  int val;
  struct node* next;
};

int main() {
  struct node n1 = {1, NULL};
  struct node n2 = {2, NULL};
  struct node n3 = {3, NULL};

  struct node* list;
  list = &n1;
  n1.next = &n2;
  n2.next = &n3;

  for (struct node* n = list; n != NULL; n = n->next) {
    printf("Val: %d\n", n->val);
  }
}
```

# Example: Dynamic Linked list (using malloc/free)

```
struct node {
  int val;
  struct node* next;
};

struct node* insert_front(int val, struct node* head)
{
  struct node* n = malloc(sizeof(struct node));
  if (n == NULL) {
    printf("ERROR: Out of space!\n");
    exit(1);
  }
  n->val = val;
  n->next = head;
  return n;
}
```

```
void print(struct node* list) {
  for (struct node* n = list; n != NULL; n = n->next) {
    printf("Val: %d\n", n->val);
  }
}
```

```
int main() {
  struct node* n3 = insert_front(2, NULL);
  struct node* n2 = insert_front(1, n3);
  struct node* n1 = insert_front(0, n2);

  free(n1);
  free(n2);
  free(n3);
}
```

# Visualize: insert_front

Assume insert_front is called twice, like so:

```
struct node* n2 = insert_front(1, NULL);
struct node* n1 = insert_front(0, n2);
```

```
struct node {
  int val;
  struct node* next;
};

struct node* insert_front(int val, struct node* head)
{
  struct node* n = malloc(sizeof(struct node));
  if (n == NULL) {
    printf("ERROR: Out of space!\n");
    exit(1);
  }
  n->val = val;
  n->next = head;
  return n;
}
```

# Visualize: print

```
void print(struct node* list) {
  for (struct node* n = list; n != NULL; n = n->next) {
    printf("Val: %d\n", n->val);
  }
}
```

Assume the list contains the numbers 0, 1, and 2.

# Exercise: Visualize this program

```
void print(struct node* list) {
  for (struct node* n = list; n != NULL; n = n->next) {
    printf("Val: %d\n", n->val);
  }
}
```

Assume we have a linked list with 3 nodes. The first node has value 10, the second has value -3, and the third has value 5. Visualize the execution of `print`

# Exercise: Draw the stack diagram

```
struct node* insert_front(int val, struct node* head)
{
  struct node* n = malloc(sizeof(struct node));
  if (n == NULL) {
    printf("ERROR: Out of space!\n");
    exit(1);
  }
  n->val = val;
  n->next = head;
  return n;
}
```

```
int main() {
   struct node* n1 = insert_front(0, NULL);

   free(n1);
   // draw the stack diagram here
}
```

# Example: Dynamic linked list – version 2

```
struct node {
  int val;
  struct node* next;
};

void insert_front(int val, struct node* head) {
  struct node* n = malloc(sizeof(struct node));
  if (n == NULL) {
    printf("ERROR: Out of space!\n");
    exit(1);
  }
  n->val = val;
  n->next = head;
  head = n;
}
```

```
void print(struct node* list) {
  for (struct node* n = list; n != NULL; n = n->next) {
    printf("Val: %d\n", n->val);
  }
}

int main() {
  struct node* head = NULL;
  insert_front(0, head);
  insert_front(1, head);
  insert_front(2, head);
  print(head);
}
```

This doesn’t work. Why?

# Example: Dynamic linked list – version 2

```
void insert_front(int val, struct node* head) {
  struct node* n = malloc(sizeof(struct node));
  if (n == NULL) {
    printf("ERROR: Out of space!\n");
    exit(1);
  }
  n->val = val;
  n->next = head;
  head = n;
}

int main() {
  struct node* head = NULL;
  insert_front(0, head);
  insert_front(1, head);
  print(head);
}
```

# Example: Dynamic linked list – version 2 (fixed)

```
void insert_front(int val, struct node** head) {
  struct node* n = malloc(sizeof(struct node));
  if (n == NULL) {
    printf("ERROR: Out of space!\n");
    exit(1);
  }
  n->val = val;
  n->next = head;
  *head = n;
}

int main() {
  struct node* head = NULL;
  insert_front(0, &head);
  insert_front(1, &head);
  print(head);
}
```

# Data Structure design in C

Summary: Use structs to encapsulate data and functions to perform operations

Exercise: Design a struct for a re-sizable string type. What functions would you need?

# Aside: Writing C programs with multiple files

A **header** file includes definitions for structs, constants, and functions so that they can be re-used

Examples: stdio.h, stdlib.h

Define your own headers for your own code:

```
#include “myfile.h”
```

Demo: mystring.h

```
// Example header file structure
#ifndef _filename_H_ // define a macro to avoid redefinitions
#define _filename_H_

// define structs and constants
#define ARRAY_SIZE 32
struct dataT
{
   int x;
   char buffer[ARRAY_SIZE];
};

// use extern for functions
extern void init_data(struct dataT* data);

#endif // close #ifdef
```

# Exercise: Binary tree

How would you implement a binary tree in C? What structs would you need? What functions would you need?

# Getting started with your own editor

The ncurses library

Content vs. views

Editing text

# ncurses

Library for drawing text in the terminal

```
#include <ncurses.h>

int main()
{
   initscr();        /* Start curses mode        */
   printw("Hello World !!!");  /* Print Hello World        */
   refresh();        /* Print it on to the real screen */
   getch();          /* Wait for user input */
   endwin();         /* End curses mode       */

   return 0;
}
```

# ncurses concepts

The **screen** represents the terminal where text can is displayed

(0,0) is the top, left corner

(num_cols, num_rows) is the bottom, right corner

```
int num_rows, num_cols;
getmaxyx(stdscr, num_rows, num_cols);
```

The **cursor** controls where text is read or written

```
move(0, 0);  //moves cursor
getyx(stdscr, r, c);  // get cursor pos
addch('a'); // display 'a' at cursor
mvaddch(10, 10, 'b'); // move cursor to (10,10) and display 'b'
```

```
initscr();
keypad(stdscr, TRUE); // enable keyboard arrows
cbreak(); // disable line buffering, e.g. trigger key input right away
noecho(); // don't show typed input automatically

int r, c;  int timeToQuit = 0;
while(!timeToQuit)
{
   int ch = getch();
   getyx(stdscr, r, c);

   if (ch == ESCAPE) timeToQuit = 1;
   else if (ch == KEY_UP) move(r - 1, 0);
   else if (ch == KEY_DOWN) move(r + 1, 0);
   else if (ch == KEY_LEFT) move(r, c - 1);
   else if (ch == KEY_RIGHT) move(r, c + 1);
   else if (ch == KEY_BACKSPACE) { move(r, c-1); addch(' '); move(r, c-1); }
   else addch(ch);
   refresh();
}
endwin();
```

# Example: Simple typing demo

# Example: reading command

```
char mesg[]="Enter a string: ";       /* message to be appeared on the screen */
char str[80];
int row,col;            /* to store the number of rows and *
                         * the number of colums of the screen */
initscr();              /* start the curses mode */

getmaxyx(stdscr,row,col);     /* get the number of rows and columns */
mvprintw(row/2,(col-strlen(mesg))/2,"%s",mesg);

getstr(str);   //  read characters into str until the user presses ENTER
mvprintw(LINES - 2, 0, "You Entered: %s", str);
getch();
endwin();
```

# Example: setting colors

```
initscr();
keypad(stdscr, TRUE); // enable keyboard arrows
cbreak(); // disable line buffering, e.g. trigger key input right away
noecho(); // don't show typed input automatically
start_color();
init_pair(1, COLOR_WHITE, COLOR_BLUE);
init_pair(2, COLOR_RED, COLOR_BLACK);

int r, c;
while(!timeToQuit)
{
   getmaxyx(stdscr,r,c);
   attron(COLOR_PAIR(2));
   mvprintw(10, 0, "Viola !!! In color ...");
   attroff(COLOR_PAIR(2));
   mvprintw(r-1, 0, "Viola !!! In color ...");
   mvchgat(r-1, 0, -1, A_NORMAL, 1, NULL);

   move(0,0);
   printw("Viola !!! In color ...");
   refresh();
}
endwin();
```

# ncurses: handling files

```
.war and peace .
.by leo tolstoy/
tolstoi ......bo
ok one 1805 ....
..chapter i ...w
ell prince so ge
noa and lucca ar
e now just famil
y estates of the
 .buonapartes bu
t i warn you if
you dont tell me
 that this means
 war .if you sti
ll try to defend
 the infamies an
d horrors perpet
rated by that .a
ntichrist  i rea
lly believe he i
s antichrist  i
```

Text files contain a sequence of characters

Editors display text line by line

We need to read the text into a data structure that supports line-by-line editing

For large files, you can only show a portion of the file in the terminal

# ncurses viewport handling

The file has lines from 0 to N where N is the number of lines

The terminal is a box that ranges from (0,0) in the top left and (terminal_height, terminal_width) in the bottom right.

To display text, we need to write to the terminal

- From (0,0) to (terminal_height, terminal_width)
- with the file contents from
  - (current_start_row, 0) to
  - (current_start_row + terminal_height, current_start_col + terminal_width)