Problem Solving using Structures

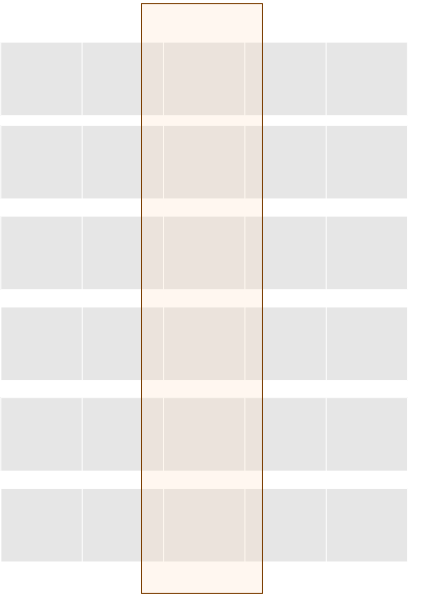
Rupesh Nasre.

Mentor TAs: Ahmed, Vivek, Vimala, Akash,
Kankan, Rahul, Swati, Akshay, Ashok, Keshav

IIT Madras
May 2022

Problem: Maintain Student Records

- A student's information consists of:
  - Roll number (key / unique id)
  - Name
  - Hostel name
  - Room number (within the hostel)
  - Date of birth
  - Mobile number
  - ...

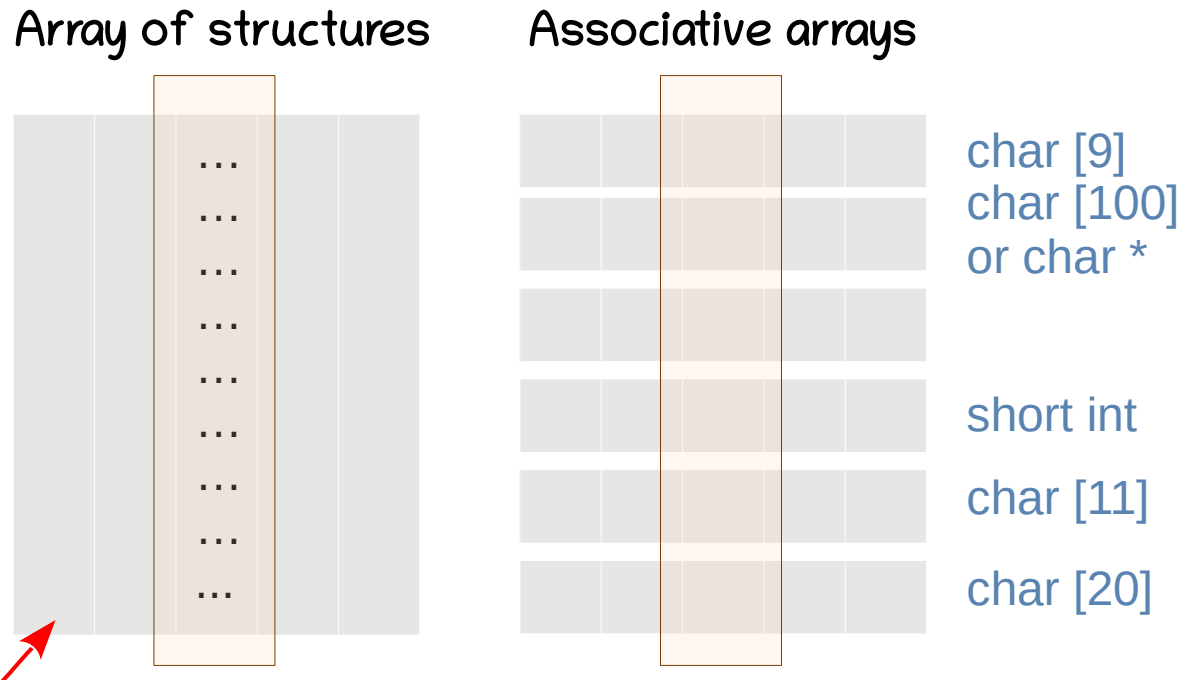


- Functions to maintain this information
  - int index = **create**("CS21B001", "Aadyot Bhardwaj", "Mahanadi", 213, "10/09/2003", "9815921001", ...);
  - **findinfo**("CS21B001", name, hostel, &room, dob, mobile, ...);
  - **update**(index, rollno, name, hostel, room, dob, mobile, ...);

Problem: Maintain Student Records

```
struct student {
        char rollno[9];
        char name[100];
        char hostel[20];
        short int room;
        ...
};

struct student thisstudent;
struct student allstuds[87];
```



- Functions to maintain this information
  - int index = **create**(thisstudent);
  - **findinfo**(index, &thisstudent);
  - **update**(index, thisstudent);

  - int index = **create**("CS21B001", "Aadyot Bhardwaj", "Mahanadi", 213, "10/09/2003", "9815921001", ...);
  - **findinfo**("CS21B001", &thisstudent);
  - **findinfo**("CS21B001", name, hostel, &room, dob, mobile, ...);
  - **update**(index, rollno, name, hostel, room, dob, mobile, ...);

struct Examples

```
struct student {
        char rollno[9];
        char name[100];
        char hostel[20];
        short int room;
        ...
};

struct student thisstudent;
struct student allstuds[87];
```

```
struct oneCourse {
        char title[7];
        int ncredits, points;
};
struct myRecord {
        struct oneCourse perf[10];
        float cgpa;
};
```

```
struct point {
        double x, y;
};
struct rectangle {
        struct point lowleft, topright;
};
struct square {
        struct point lowleft;
        unsigned side;
};
struct rectangle *boxes;
struct square *sqboxes[N];
```

These are only declarations. No memory is allocated.

These are variables. Memory is allocated.

```
struct oneDish {
        char *item;
        int quantity;
};
typedef struct oneDish dishes[2];
dishes birthday, job, graduation;
```

New type

New name to a type
Variables

struct Use

```
struct oneDish swiggy;
swiggy.item = "Pizza";
swiggy.quantity = 4;

job[0].quantity = 50;
job[0].item = strdup("Samosa");
printf("sizeof(job) = %lu\n", sizeof(job));

(*birthday).quantity = 1;
birthday[0].item = "Heart shaped cake";
puts((*birthday).item);
```

```
struct oneDish {
        char *item;
        int quantity;
};
typedef struct oneDish dishes[2];
dishes birthday, job, graduation;
```

New type

New name to a type

Variables

More Examples

Declare a structure for a library book.
Create a variable to hold information about Ritchie‘s book.
Create a variable to hold information for ten C related books.
Set the title of Ritchie‘s book.
Find a C book from McGraw Hill publishers.

Analyze the code.

```
struct s {
        int x, y;
        char a[10], b[10];
        char *c, *d;
} s1, s2;
void populate(struct s s1) {
        s1.x = 1;
        s1.y = 2;
        strcpy(s1.a, "Hello");
        strcpy(s1.b, "World");
        s1.c = (char *)malloc(10);
        strcpy(s1.c, "Bye");
        s1.d = (char *)malloc(10);
        strcpy(s1.d, "Heaven");

}

int main() {
        populate(s1);
        s2 = s1;
}
```

Problem: Sort student records.

- Recall our sorting algorithms.

```
for (ii = 0; ii < N - 1; ++ii)
    for (jj = 0; jj < N - ii - 1; ++jj)
        if (arr[jj] > arr[jj + 1]) swap(arr[jj], arr[jj + 1]);
```

&arr[jj], &arr[jj + 1] OR
arr, jj, jj+1

We need to specify how to compare two records.

```
void swap(struct s arr[], int x, int y) {
    struct s tmp = arr[x];
    arr[x] = arr[y];
    arr[y] = tmp;
}
```

✓

```
void swap(struct s *x, struct s *y) {
    struct s *tmp = x;
    x = y;
    y = tmp;
}
```

```
void swap(struct s *x, struct s *y) {
    struct s tmp = *x;
    *x = *y;
    *y = tmp;
}
```

✓

Problem: Sort student records.

```
struct student {
        char rollno[9];
        char *name;
        short int room;
        ...
};
```

We need to specify how to compare two records.

Write the code to sort by name.

```
int compareByRoom(struct s arr[], int ii, int jj) {
        return (arr[ii].room > arr[jj].room);
}
void sortByRoom(struct s arr[], int N) {
        for (int ii = 0; ii < N - 1; ++ii)
                for (int jj = 0; jj < N - ii - 1; ++jj)
                        if (compareByRoom(arr, jj, jj+1))
                                swap(arr, jj, jj+1);
}
```

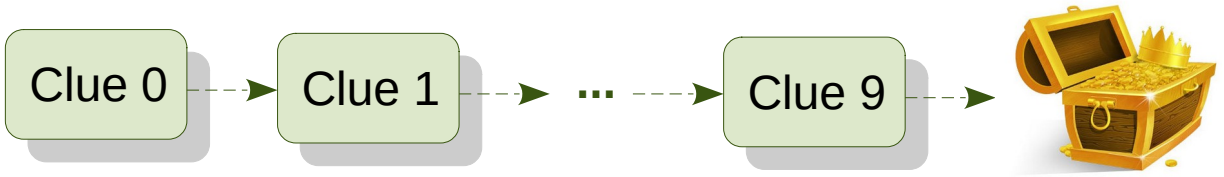
```
int compareByName(struct s arr[], int ii, int jj) {
        return (strcmp(arr[ii].name, arr[jj].name) > 0);
}
void sortByName(struct s arr[], int N) {
        for (int ii = 0; ii < N - 1; ++ii)
                for (int jj = 0; jj < N - ii - 1; ++jj)
                        if (compareByName(arr, jj, jj+1))
                                swap(arr, jj, jj+1);
}
```


Homework: Perform binary search by room, by name.

Source: sortstruct.c

Issues with Arrays

- Sometimes, not all the students are present.
- Sometimes, data is getting streamed.
- Creating `struct student arr[N];` can waste space.
- It would be nice if
  - memory is allocated as and when the datum arrives
  - memory is allocated only for the arrived data
- Need to compromise with contiguousness.



Linked List

```
struct student {
        char rollno[9];
        char name[100];
        char hostel[20];
        short int room;
        ...
        struct student *next;
};
```

```
void print(struct student *head) {
        struct student *ptr = head;
        while (ptr != NULL) {
           printf("%s %s\n", ptr->rollno, ptr->name);
           ptr = ptr->next;
        }
}
```

(*ptr).field is so common that C has a shorthand for it: ptr→field written as ptr->field.

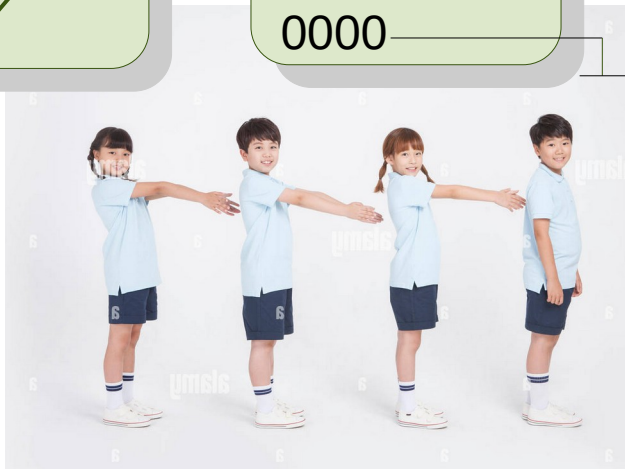
head

1924

1924	2044	2084	1096	1152
CS21B008 Arjun … 2044	CS21B084 Ujjayan … 2084	CS21B051 Mayank … 1096	CS21B035 Isha … 1152	CS21B010 Arun … 0000

If I know the address 1924, I can access all the data in the list.

Clue 0 --> Clue 1 --> ... --> Clue 9 -->



Let's create a linked list.

Zero nodes

```
struct student *head = NULL;
```

One node

```
void populate(struct student *ptr) {
      scanf("%s%s%s%d",  ptr->rollno, ptr->hostel,
                         ptr->name, &ptr->room);
       ptr->next = NULL;
}
...
struct student *head = NULL;

head = (struct student *)malloc(sizeof(struct student));
populate(head);
```

Many nodes

```
void populate(struct student *ptr) {
      scanf("%s%s%s%d",  ptr->rollno, ptr->hostel,
                         ptr->name, &ptr->room);
       ptr->next = NULL;
}
...
struct student *head = NULL;
struct student **ptr = &head;
for (int ii = 0; ii < N; ++ii) {
      *ptr = (struct student *)malloc(sizeof(struct student));
      populate(*ptr);
      ptr = &((*ptr)->next);
}
```

```
void populate(struct student *ptr) {
        scanf("%s%s%s%d",  ptr->rollno, ptr->hostel,
                                ptr->name, &ptr->room);
        ptr->next = NULL;
}
...
struct student *head = NULL;
struct student **ptr = &head;
for (int ii = 0; ii < N; ++ii) {
        *ptr = (struct student *)malloc(sizeof(struct student));
        populate(*ptr);
        ptr = &((*ptr)->next);
}
```

Can we avoid pointer to a pointer?

Many nodes

```
...
struct student *head = populate();
struct student *ptr = head;
for (int ii = 1; ii < N; ++ii) {
        ptr->next = populate();
        ptr = ptr->next;
}
```

```
...
struct student *head = NULL;
struct student *ptr;
for (int ii = 0; ii < N; ++ii) {
        if (ii == 0) {
                head = populate();
                ptr = head;
        } else {
                ptr->next = populate();
                ptr = ptr->next;
        }
}
```

Printing a list

Try printing the list in reverse.
- iterative vs. recursive

```
void print(struct student *ptr) {
        while (ptr != NULL) {
            printf("%s %s\n", ptr->rollno, ptr->name);
            ptr = ptr->next;
        }
}
```

iterative

Modify the code to support searching for an element.

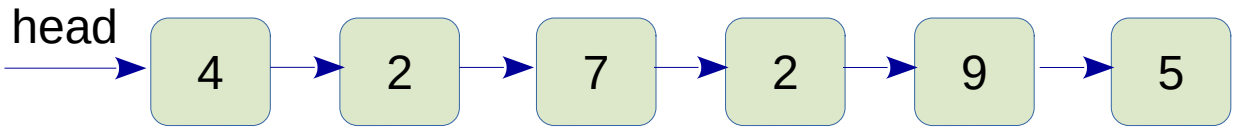
```
void printRec(struct student *ptr) {
        if (ptr != NULL) {
            printf("%s %s\n", ptr->rollno, ptr->name);
            printRec(ptr->next);
        }
}
```

recursive

Deleting from a list

- remove(2)
- remove(5)
- remove(4)

We want to remove all occurrences of the value.



Special case:

```
if (head == NULL) return;
```

General case:

```
struct node *previous = NULL;
for (struct node *ptr = head; ptr;) {
        if (ptr->val == val) {
                struct node *toberemoved = ptr;
                if (previous) {
                        previous->next = ptr->next;
                } else head = ptr->next;
                ptr = ptr->next;
                free(toberemoved);
        } else {
                previous = ptr;
                ptr = ptr->next;
        }
}
```

Pitfalls

- ptr = head->next; // segfault. Check if head is NULL.
- node *ptr = &node1; return; // local variable node1.
- ptr = malloc(sizeof(node*)); // insufficient memory.

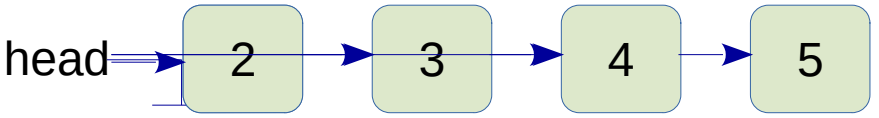
Wrong **deleteList** program

```
for (ptr = head; ptr; ptr = ptr->next)
    free(ptr);                          // invalid memory on free.
                                        // may work but wrong.
```

Problem: Bank token system

- As customers enter bank, they should be issued a numeric token (1, 2, 3, ...).
- Customers are served in the token order.
- If we constraint our linked list as:
  - Insertions happen at one end
  - Removals happen at the other end

New customer arrives
Serve
New customer arrives
New customer arrives
Serve
New customer arrives
New customer arrives
Serve



Given a sequence of N (new customer) and S (serve), print the customer tokens in the queue at each stage.

Problem: Bank token system

Declarations in q.h

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
        int token;
        struct Node *next;
};
typedef struct Node *Tokens;
typedef char Event;
enum {Exit = 'E',
          NextCustomer = 'N',
          Serve = 'S'};
```

Queue processing in q.c

```
#include "q.h"

void printTokens(Tokens t) {
        …
}
void genNextToken(Tokens *t) {
        …
}
void serveCustomer(Tokens *t) {
        …
}
```

main in qmain.c

```
#include "q.h"

extern void printTokens(Tokens);
extern void genNextToken(Tokens *);
extern void serveCustomer(Tokens *);

int main() {
        ...
}
```

Compile

```
gcc q.c qmain.c
```

OR

```
gcc -c q.c
gcc -c qmain.c
gcc q.o qmain.o
```

```
New customer arrives
Serve
New customer arrives
New customer arrives
Serve
New customer arrives
New customer arrives
Serve
```

Makefile

```
a.out: qmain.o q.o
        gcc qmain.o q.o

qmain.o: qmain.c q.h
        gcc -c qmain.c

q.o: q.c q.h
        gcc -c q.c
clean:
        rm a.out q.o qmain.o
```

head → 4 → 5

Compile with Makefile

```
make clean
make
rm q.o
make q.o
touch q.h
make
touch q.c
make
```

Given a sequence of N (new customer) and S (serve), print the customer tokens in the queue at each stage.

union

```
#include <stdio.h>

union Node {
        char event;
        int token;
        float entity;
};

int main() {
        union Node n;
        printf("%p %p %p\n", &n.token, &n.event, &n.entity);
        n.token = 65;
        printf("%d = %c\n", n.token, n.event);
}
```

	Week	Problems	Tools
✓	0	Solve equations, find weighted sum.	Data types, expressions, assignments
✓	1	Find max, convert marks to grade.	Conditionals, logical expressions
✓	2	Find weighted sum for all students.	Loops
✓	3	Encrypt and decrypt a secret message.	Character arrays
✓	4	Our first game: Tic-tac-toe	2D arrays
✓	5	Making game modular, reuse.	Functions
✓	6	Find Hemachandra/Fibonacci numbers.	Recursion
✓	7	Encrypt and decrypt many messages.	Dynamic memory, pointers
✓	8	Maintain student records.	Aggregate data types
✓	9	Search and sort student records.	Searching and sorting algorithms
✓	A	Reduce memory wastage.	Linked lists
✓	B	Implement token system in banks.	Queues
	C	IRCTC-like ticket booking system	File handling
	D	Putting it all together	All the above