PROBLEM SOLVING USING C

UNIT-3

PRIORITISED SCHEDULING

Vibha Masti

Feedback/corrections: vibha@pesu.pes.edu

Multi-Dimensional Array

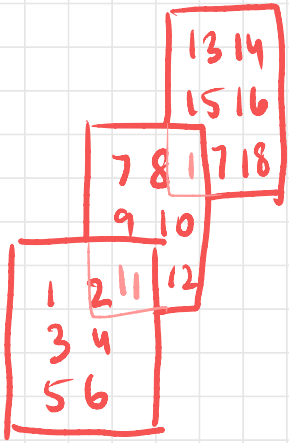
(only integer data)

int A[3][3][2] ⟶ 18 entries

```
{{{1,2},{3,4},{5,6}},
 {{7,8},{9,10},{11,12}},
 {{13,14},{15,16},{17,18}}}
```

(2 · 3 · 3)

3 × 3 × 2



looping and displaying

```
int A[3][3][3];

int m, n, o;

scanf("%d %d %d", &m, &n, &o); // m=2, n=2, o=3


for(int i=0; i<m; ++i) {
    for(int j=0; j<n; ++j) {
        for(int k=0; k<o; ++k) {
            :
        }
    }
}
```

Replace 'i' with 'e'

```
char str[] = "Twinkle twinkle little star";

char *cp = str;

for(int i=0; *(cp+i); ++i) {

    if(*(cp+i) == 'i') {
        *(cp+i) = 'e';
    }

}
```

Heterogeneous Data Structure

Student

name - str	branch - str
ID - str	cgpa - float
sex - char	email - str
SRN - str	phone - long
DOB - date	

different data types in a structure

related fields grouped together in a record

Structure

```
struct new_datatype {

        datatype 1   variable 1;
        datatype 2   variable 2;
              :

        datatype n   variable n;

};
```

← declaring a new var. of type new_datatype

```
struct new_datatype instance 1;
```

eg:
```
struct Student {
    char *name, *id, *srn, *branch, *email;
    char sex;
    float cgpa;
    long phone;
    Date dob;
};
```

assume another struct

NOTE: no memory is allocated while defining a structure.

Only when a variable is declared, memory is allocated while compiling

Total size of structure variables = sum of size of individual variables inside struct

sizeof char* = 4 (address)

Accessing Attributes

```
struct customer {
    int custid;          ——— 4
    char *custname;      ——— 4
    float itemprice;     ——— 4
};                           12

struct customer c;
    :
    :
c.custid;  ⟶ returns value of custid in variable c
c.custname;
c.itemprice;
```

NOTE

```
struct Customer {
    int custid;          ——— 4
    char custname[50];   ——— 50
    float itemprice;     ——— 4
};                           58
```

Array of Structures

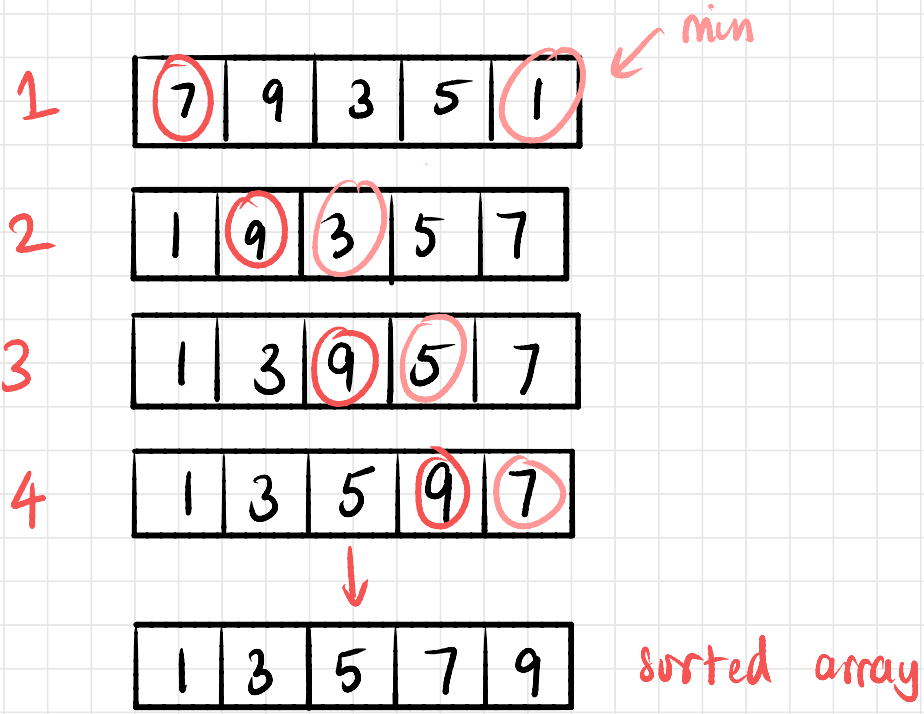
```
struct customer {
    int custid;
    char *custname;
    float item price;
};

struct customer customers[30];
```

array of type customer

Sorting

Selection Sort

- find min el & position
- swap with first element



Code

```
void selsort(int arr[], int n) {

    int min, ind, temp;

    for (int i=0; i<n; ++i) {
        min = arr[i];
        ind = i;
        for(int j=i+1; j<n; ++j) {
            if (arr[j] < min) {
                min = arr[j];
                ind = j;
            }
        }

        if (i != ind) {
            temp = arr[i];
            arr[i] = min;
            arr[ind] = temp;
        }
}
```

outer loop

inner loop

swap the variables

Typedef (Alias)

```
typedef <known datatype> <unknown datatype>;

typedef float MONEY;

MONEY sal;  ⟶ equivalent to float sal;

typedef char[30] str;
str name;
```

On structs

```
type def  struct  customer {
            int custid;
            char *custname;
            float itemprice
} cust;
```

NOT A VARIABLE; AN ALIAS

```
cust c1;  ⟶ same as customer c1;
```

Sorting an array of structures

- sort normally

```
void selsort(struct customer A[], int n) {

    int min, i, j;

    for(i=0; i<n-1; ++i) {
        min = i;
        for (j=i+1; j<n; ++j) {
            if (A[j].id < A[min].id) {
                min = j;
            }
        }
        struct customer temp = A[i];
        A[i] = A[min];
        A[min] = temp;

    }
```

Dynamic Memory Allocation

```
struct bank_acc {
    int accno;
    char name[STR_LEN];
    char type;
    float amount;
};
```

```
struct bank_acc b1[1000];
```

static memory ← (waste of space or lack of space)

static memory → allocated at runtime

- static memory: underutilisation or lack of sufficient memory
- today, memory not a concern
- discrete allocation
- dynamic memory allocation: continuous allocation.

FUNCTIONS FOR DMA

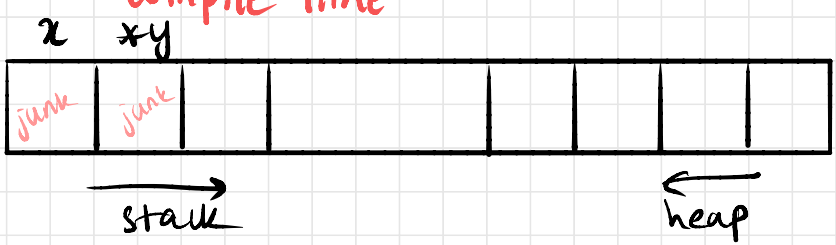
1. malloc (parameter) → stdlib.h

   ↑ locations

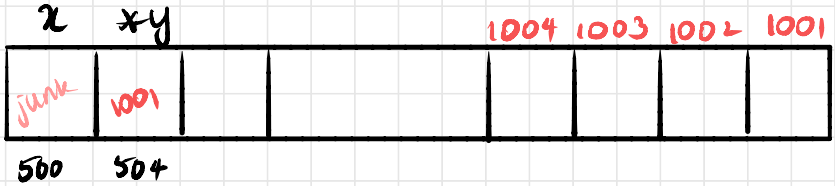
```
int x;
int *y;
y = (int*) malloc (sizeof(int));
```

← returns void *

compile time



runtime



2.
3.

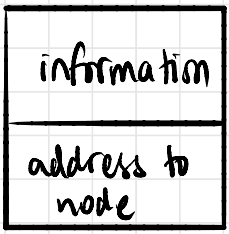
LINKED LIST

logically sequential

self-referential structure

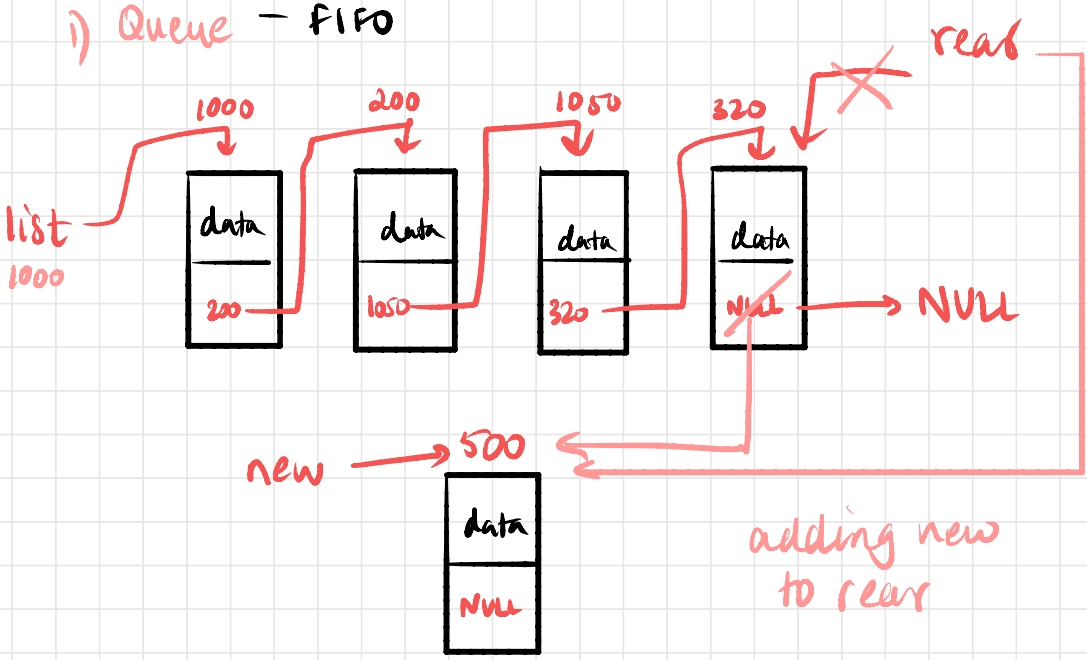
```
struct node {
    int data;
    struct node *next;
};
```

node

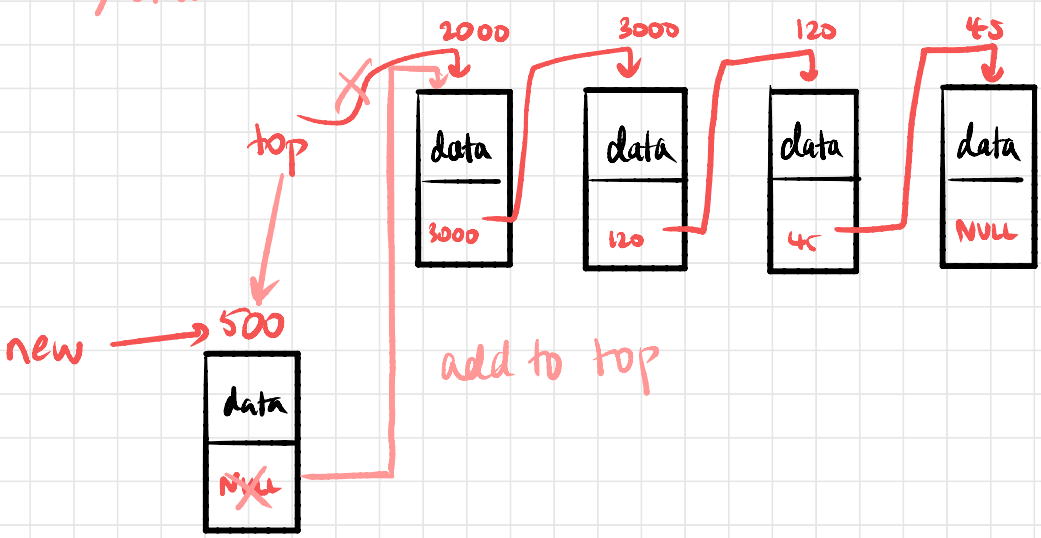


Creating a Linked List

1) Queue – FIFO



2) Stack – LIFO



Stack

```
struct node *new = NULL, *list = NULL;

void push(int x) {
    new = (struct node*) malloc(sizeof(struct node));

    new -> data = x;
    new -> next = NULL;

    if (list == NULL) {
        list = new;
    }

    else {
        new -> next = list;
        list = new;
    }

}
```

typecast

carries address of first node