Systems Software/Programming

C Programming Revision

Why C for Systems Software/Programming ?

- Unix/Linux OS has been written in C/C++, hence call to system functions is easier in C.
- C can perform memory management through dynamic memory allocation.
- C can implement different data structures efficiently using structures and pointers such as stack, queues, linked list, trees etc.
- Using C, hardware systems can be programmed directly.
- C can call assembly routines and assembly code can call C routines on many processors if required.

C – Primitive Data types

- char : character - 1 byte
- short: short integer - 2 bytes
- int: integer - 4 bytes
- long: long integer - 4 bytes
- float: floating point - 4 bytes
- double - double precision floating point - 8 bytes

- %d: integers
- %f: floating point
- %c: characters
- %s: string
- %x: hexadecimal
- %u: unsigned int

Operators

- Arithmetic: + - / * %
- Relational: > >= < <= == !=
- Logical: && || !
- Increment and Decrement: ++ and -- (pre and post)
- Bitwise: & | ^ << >> ~
- Assignment: = += -= /= *= %= &= |= ^= <<= >>=
- Ternary: ?:
- Special: .(dot) -> (structure member access using pointer)

Constants, Macros and Variables

```
#define COURSE “Systems Programming”
#define NO_OF_STUDENTS 120
#define max(a,b) a>b?a:b
```

```
int i;
float f = 10.8f;
double d = 17.8;
char c = 65;
```

C Statements

- Conditionals
  - IF, IF-ELSE, Nested IF-ELSE
  - switch..case…default
- Loops
  - for(initialization;condition;iteration)
  - while
  - do..while
  - break
  - Continue
  - goto

Arrays

- Defining an array is easy:

```
int a[3];   /* a is an array of 3 integers */
```

- Array indexes go from 0 to n−1:

```
a[0] = 2; a[1] = 4; a[2] = a[0] + a[1];
int x = a[a[0]];     /* what is the value of x? */
```

  - **Beware:** in this example a[3] does not exist, but your compiler will not complain if you use it!
    - ★ But your program may have a very strange behavior. . .

- You can create multidimensional arrays:

```
int matrix[3][2];
matrix[0][1] = 42;
```

- Arrays can be of primitive data type int, float, double, char
- Array can be of user defined data type using typedef

typedef char20 char[20];

char20 c20arr[10];

- Array can be of structure or union

struct user {

int user_id;

char[20] user_name;

};

struct user users[20];

What are the different ways to initialize the array values ?

```
int a[10]={0,1,4,6,7,12,8,9,-1,34};
char c[3]={‘a’,’b’,’c’};
int b[3][2]={{2,5},{6,10},{-1,6}};
struct user u[2]={{1,”xyz”},{2,”abc”}}
```

OR

- Similarly using loops when don’t know the input values

```
for(i=0; i<10; i++)
      scanf(“%d”, &a[i]);
for(i=0; i<3; i++) {
            for(j=0;j<2;j++)
                  scanf(“%d”, &b[i]);
      }
```

Strings
Difference between Character Array and String

- Character Array does not have a ‘\0’ (NULL) to represent the end i.e. last character is regular ASCII character

I	**n**	**t**	**r**	**o**		**2**		**p**	**r**	**o**	**g**

- String ALWAYS has ‘\0’ as a last character to represent end of string.

I	**n**	**t**	**r**	**o**		**2**		**p**	**r**	**o**	**g**	**\0**

Difference between Character Array and String - How to read user input ?

All 10 locations will have ASCII Character value

```
char c_arr[10];
for (i=0; i<10; i++)
        scanf("%c",&c_arr[i])
; OR
        c_arr[i] = getchar();
```

Only first 9 locations can be used for storing actual string characters, last one will have '\0'

```
char str[10];
scanf("%s", str); OR
gets(str);
```

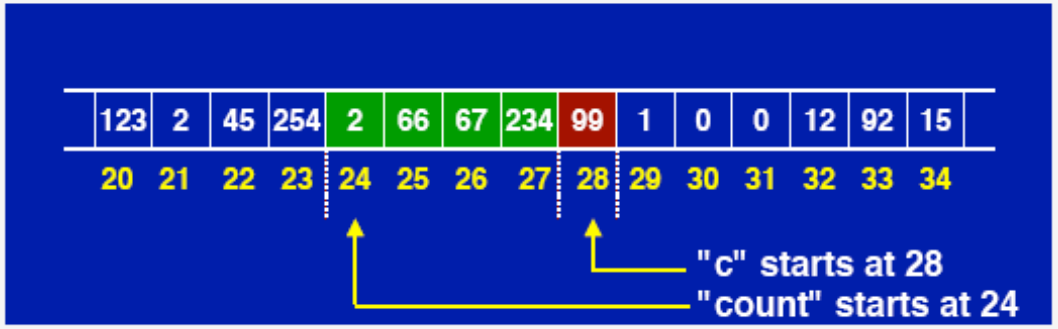
String Functions – Does not work for Character Array

- strlen(s) – returns the length of the string excluding '\0'
- strcmp(s1, s2) – returns 1 (+ve No) if s1 > s2, -1 (-ve No) if s1 < s2 and 0 if s1 equals s2
- strcpy(dst, src) – copies src string into dst string
- strcat(s1, s2) – returns concatenation of two string (i.e. s1+s2)
- strstr(lrgstr, smlstr) – index of 1st occurrence of smlstr into lrgstr

Memory Allocation for Variables

Declare variable int count and unsigned char c (what is difference between signed and unsigned value?)

- Memory is reserved to store the variables
- And the compiler 'remembers their location'



- As a result, each variable has two properties:
  1. The '**value**' stored in the variable
     - ☞ If you use the name of the variable, you refer to the variable's value
  2. The '**address**' of the memory used to store this value
     - ★ Similar to a reference in Java (but not exactly the same)
     - ☞ A variable that stores the address of another variable is called a **pointer**

- Pointers can be declared using the * character

```
int *ptr;                 /* Pointer to an int */
unsigned char *ch;        /* Pointer to an unsigned char */
struct ComplexNumber *c;  /* Pointer to a  struct ComplexNumber */
int **pp;                 /* Pointer to a  pointer to an int */
void *v;                  /* Pointer to anything (use with care!) */
```

Defining Pointers for primitive types

int i = 8, j=8;

int *p, *p1, *p2;

p = &i; p1=&i; p2=&j;

double *d = &i // will this work?

double d1 = i // will this work?

double d2=1000.0008;

i = d2; //will this work?

i = (int) d2; will this work? What (int) in this case?

sizeof(p) == sizeof(d)

sizeof(*p) == sizeof(*d)

p1 == p2

*p1 == *p2

p == p1

What will be the result in each case?

int k = 8; int *p3;

p3 = &k; *p3 = 12;

Value of k?

Note: * Has 2 uses: 1. for pointer declaration 2. for getting the data value pointed by pointer variable

C_revision\pointers_knowledge_check.c

Arrays and Pointers

```
int a[10], *ptr;
ptr= &a[0]; // ptr will point to first array element
printf("%d", *(ptr+5) );// ptr+ 5 = ?
```

If address of a[0] is 2000 then what will be the value of ptr+ 5?

```
ptr[7] = 10; // will this work?
ptr=a; // will this work?

printf("%d", *ptr++);
printf("%d", *(ptr++));
```

What is the difference?

Be aware of precedence of operators

Structures and Pointer to Structures

- You can build higher-level data types by creating structures:

```
struct Complex {
  float real;
  float imag;
};
struct Complex number;
number.real = 3.2;
number.imag = -2;

struct Parameter {
  struct Complex number;
  char description[32];
};
struct Parameter p;
p.number.real = 42;
p.number.imag = 12.3;
strncpy(p.description, "My nice number", 31);
```

Structures and Pointer to Structures

- We very often use statements like:

```
(*pointer).field = value;
```

- There is another notation which means exactly the same:

```
pointer->field = value;
```

- For example:

```
struct data {
   int counter;
   double value;
};

void add(struct data *d, double value) {
   d->counter++;
   d->value += value;
}
```

Functions: Library and User Defined

Library

- From stdio.h: printf, scanf
- From string.h: strcpy, strcmp, strstr, strlen, strcat
- From unistd.h: read, write, lseek
- From stdlib.h: open, close

User Defined

- Definition, Declaration and Call
- Scope of the variables
- Pass parameters by value or by reference (difference ?)C_revision\pass_by_value_or_ref.c
- Defined in same file vs different file
- Use of Header file: extern functions C_revision\file1_main.c, C_revision\file2_other.c, C_revision\myheader.h
- Recursion

C pointer to array function

- Declare Array of Function pointers and initialize

```
int add(int a, int b);
int sub(int a, int b);
int mul(int a, int b);
int div(int a, int b);
int (*oper[4])(int a, int b) = {add, sub, mul, div};
```

- Call the function using initialized function pointer array

```
int result = oper[i](a,b);
```

C_revision\function_ptr_example.c

Recursive Functions – Calls itself until terminating condition

```
int fibo(int num) {
    if (num == 0)    {
        return 0;
    }
    else if (num == 1)    {
        return 1;
    }
    else    {
        return(fibo(num - 1) + fibo(num - 2));
    }
}
```

Dynamic Memory Management

#include <stdlib.h>

int a[10];

int *ptr_i = (int *) malloc(10 * sizeof(int));

Are these equivalent?

You must call free() for any pointer which has been allocated a memory. What problem will it cause if you don't?

Similarly you can use pointer to structure to access the members of the structure

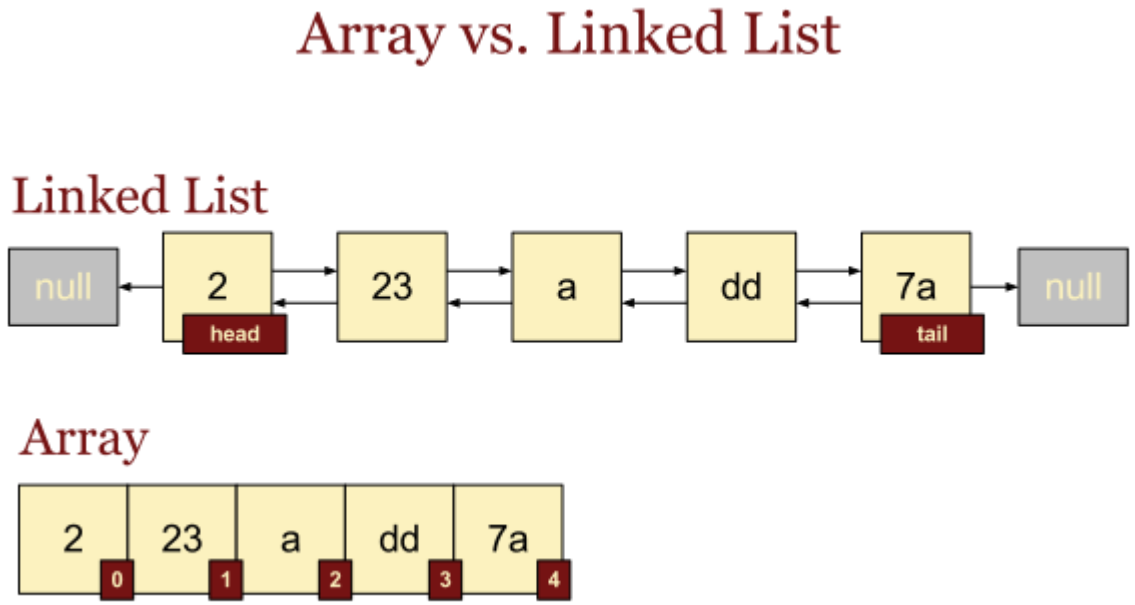
struct person { int p_id; char p_name[20];} *p

p = (struct person *) malloc(sizeof(person)); // will store only 1 person info

scanf("%d",&p->p_id); scanf("%s", p->p_name);

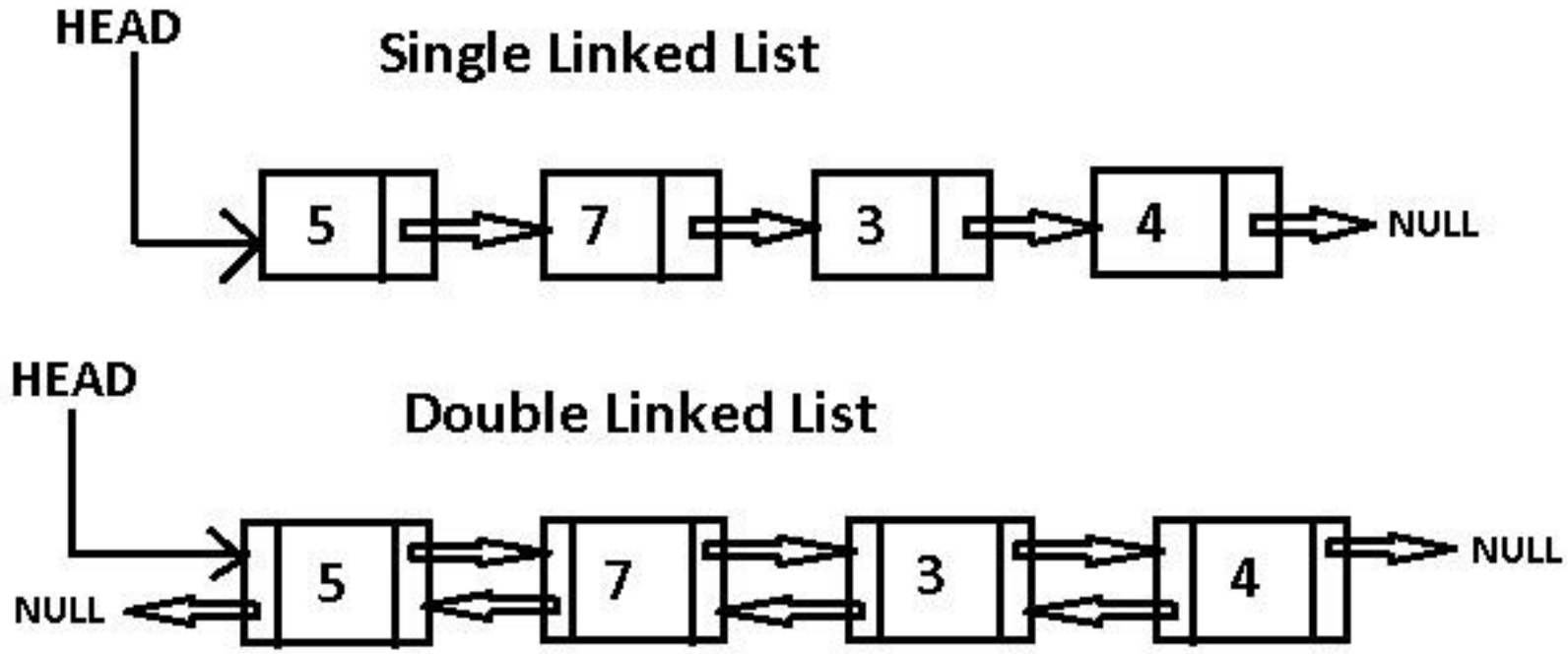
Data structures using C pointers

- Array vs Linked List



Data structures using C pointers

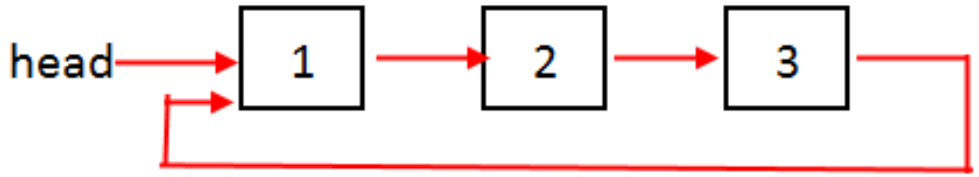
- Singly vs Doubly Linked List



Data structures using C pointers

- Singly Regular vs Singly Circular Linked List

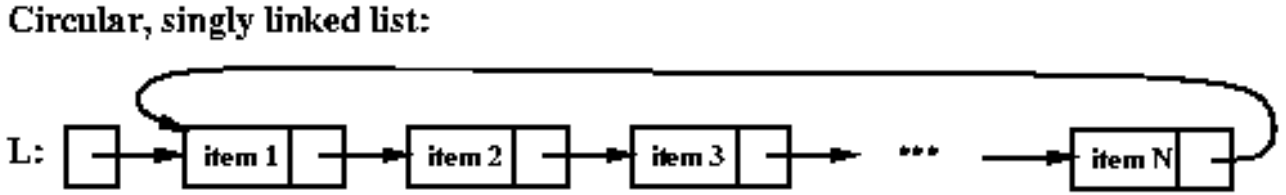
Singly Linked List

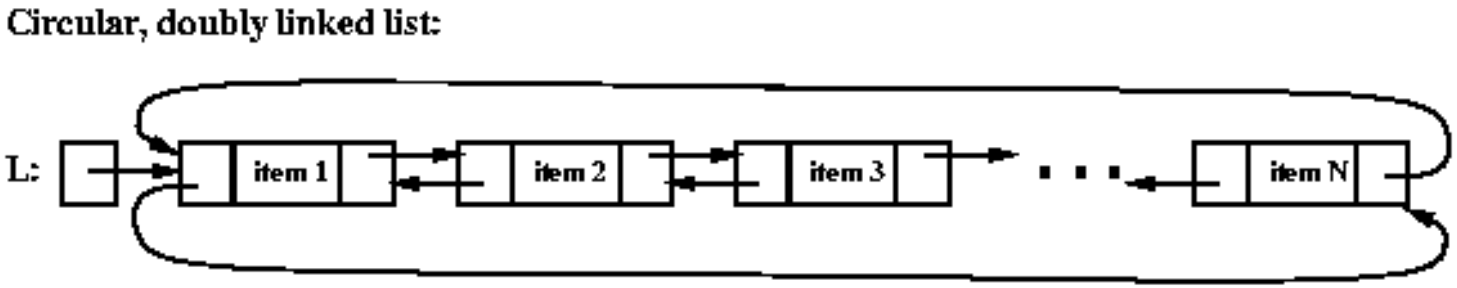


Circular Linked List

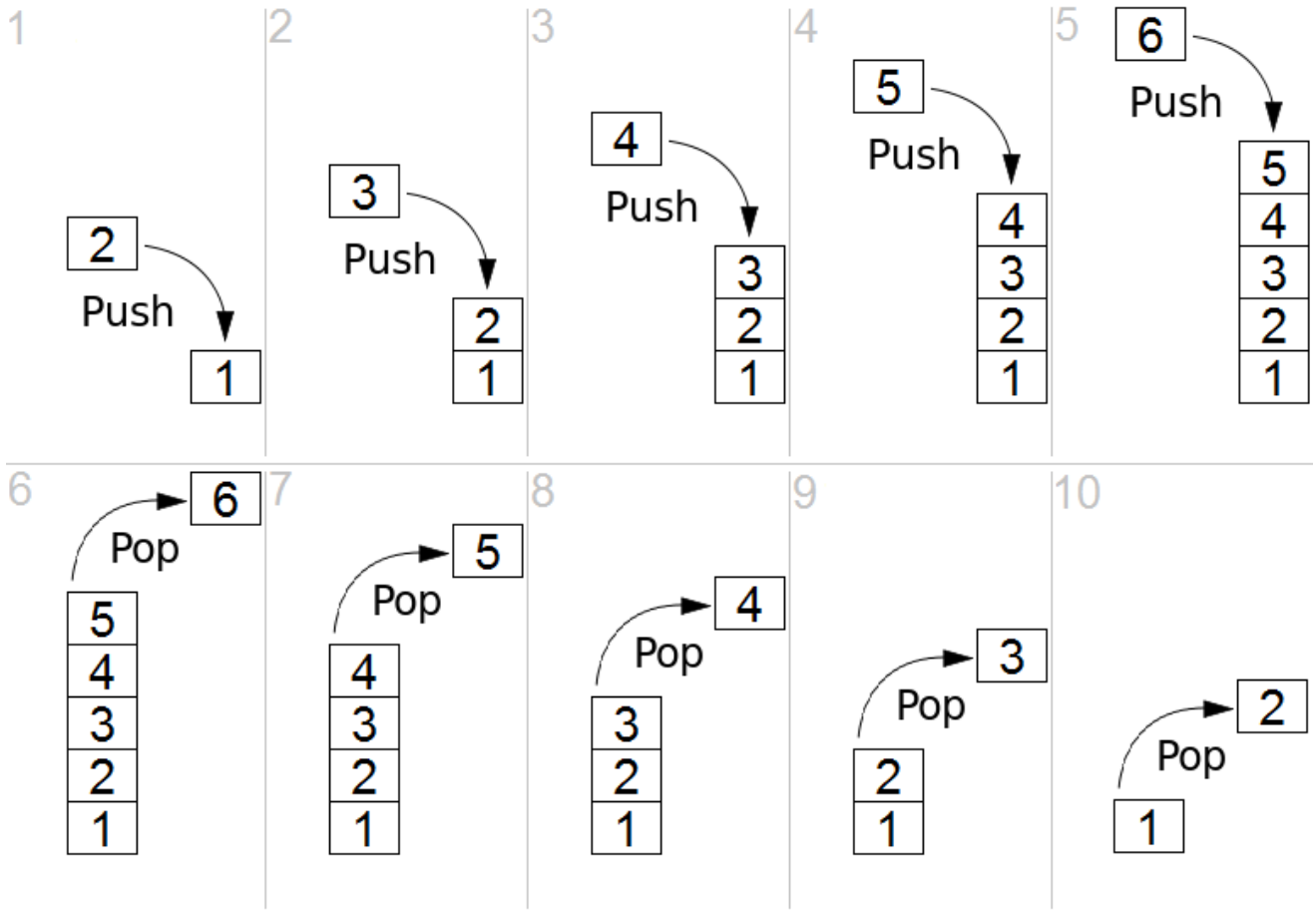
Data structures using C pointers

- Singly Circular vs Doubly Circular Linked List

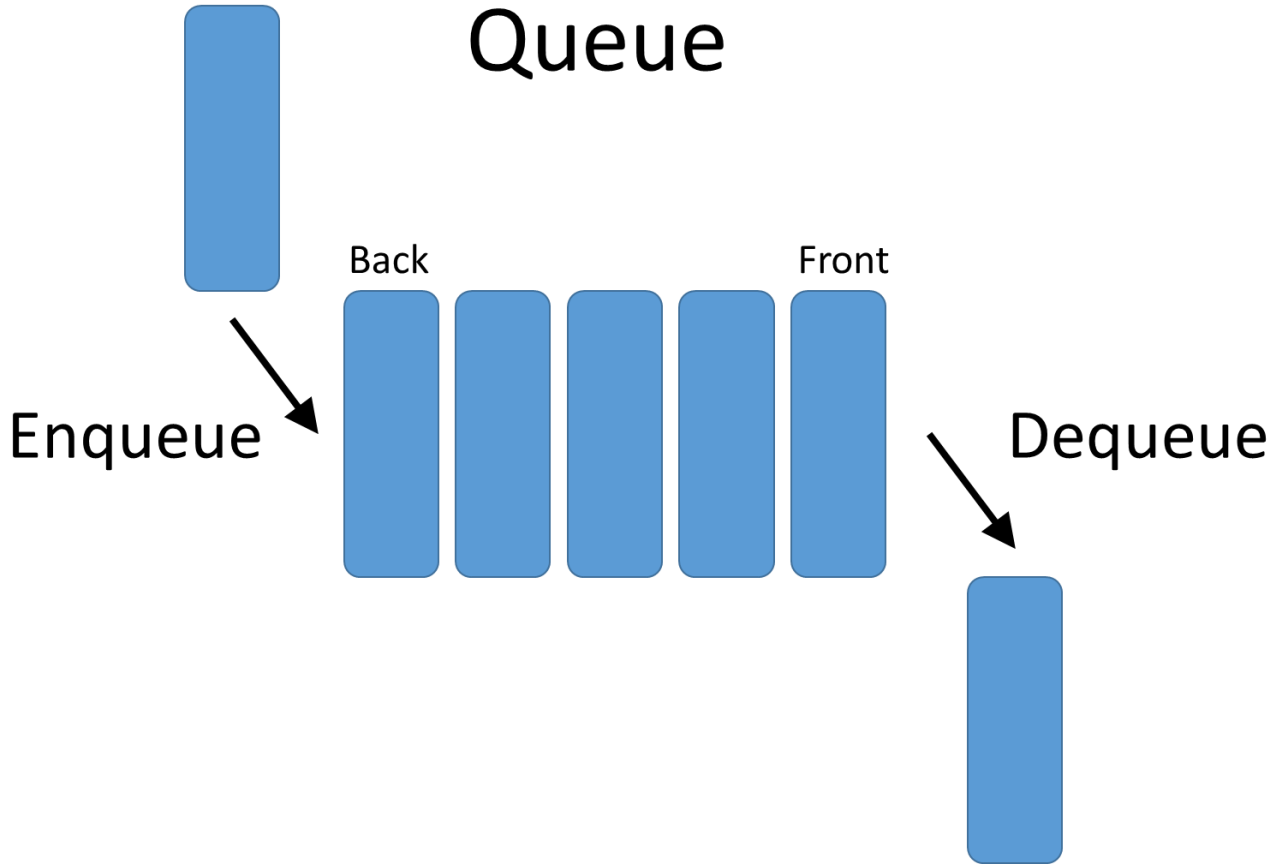




Other Data structure implementation - Stack



Other Data structure implementation - Queue



Other Data structure implementation - Trees

- Binary Trees: Binary Tree, Binary Search Tree, AVL Tree, Red-Black Tree, Splay Tree
- B-Tree: B-Tree, B+ Tree, B sharp tree
- General Trees: Trie, Radix Tree, Suffix Tree, B Trie
- Multiway Tree: Ternary Tree, K-ary Tree, And-or Tree, (a,b)-tree
- Space Partitioning Tree: K-d Tree, R-tree, Segment tree, Range tree, Octree, Herbert R-tree
- Application Specific Tree: Parse Tree, Decision Tree, MinMax Tree, Expression Tree

Example for Circular Double Linked List

```
struct user {
int user_id;
char[20] user_name;
struct user *prev;
struct user *next;
} head, tail;
```

Example for Circular Doubly Linked List

Adding a new node in the list:

```
struct user *new = (struct user *)malloc(sizeof(struct));
new->user_id=<new id>; new->user_name=<new name>;
new.next = NULL; new.prev = NULL;
if(head == NULL)
        head = new; tail = new; head->prev = head; head->next = head
else {
        tail->next = new;
        new->prev = tail;
        new->next = head;
        head->prev = new;
        tail = new;
}
```

Example for Circular Doubly Linked List

```
Delete a new node from the list for a given user_id
if(head==NULL) {
        struct user temp;
        for(temp=head; temp!=NULL; temp->next)
                if(temp->user_id == <given user_id>)
                        break;
        if(temp != NULL) {
                temp->prev->next = temp->next;
                temp->next->prev = temp->prev;
                temp->prev=NULL;
                temp->next=NULL;
                free(temp);
        }
}
```

Headers defined by the ISO (International Standard Organization) C standard

- <assert.h> : Conditionally compiled macro that compares its argument to zero
- <complex.h> (since C99) : Complex number arithmetic
- <ctype.h> : Functions to determine the type contained in character data
- <errno.h> : Macros reporting error conditions
- <fenv.h> (since C99) : Floating-point environment
- <float.h> : Limits of float types
- <inttypes.h> (since C99) : Format conversion of integer types
- <iso646.h> (since C95) _ Alternative operator spellings
- <limits.h> : Sizes of basic types
- <locale.h> : Localization utilities
- <math.h> : Common mathematics functions
- <setjmp.h> : Nonlocal jumps
- <signal.h> : Signal handling
- <stdalign.h> (since C11) : alignas and alignof convenience macros
- <stdarg.h> : Variable arguments

Headers defined by the ISO (International Standard Organization) C standard

- <stdatomic.h> (since C11) : Atomic types
- <stdbool.h> (since C99) : Boolean type
- <stddef.h> : Common macro definitions
- <stdint.h> (since C99) : Fixed-width integer types
- <stdio.h> : Input/output
- <stdlib.h> : General utilities: memory management, program utilities, string conversions, random numbers
- <stdnoreturn.h> (since C11) : noreturn convenience macros
- <string.h> : String handling
- <tgmath.h> (since C99) : Type-generic math (macros wrapping math.h and complex.h)
- <threads.h> (since C11) : Thread library
- <time.h> : Time/date utilities
- <uchar.h> (since C11) : UTF-16 and UTF-32 character utilities
- <wchar.h> (since C95) : Extended multibyte and wide character utilities
- <wctype.h> (since C95) : Functions to determine the type contained in wide character data

Required Headers for IEEE POSIX (Portable Operating System Interface) Standard (This is in addition to ISO C Standard)

- <direct.h> : directory entries
- <fcntl.h> : file control
- <fnmatch.h> : filename-matching types
- <glob.h> : pathname-pattern matching types
- <grp.h> : group file
- <netdb.h> : network database operation
- <pwd.h> : password file
- <regex.h> : regular expressions
- <tar.h> : tar archive values
- <termios.h> : terminal I/O
- <unistd.h> : symbolic constants
- <utime.h> : file times
- <wordexp.h> : word-expansion types

Required Headers for IEEE POSIX (Portable Operating System Interface) Standard (This is in addition to ISO C Standard)

- <arpa/inet.h> : Internet definition
- <net/if.h> : socket local interface
- <netinet/in.h> : internet address family
- <netinet/tcp.h> : Transmission Control Protocol definition
- <sys/nman.h> : memory management declaration
- <sys/select.h> : select function
- <sys/socket.h> : socket interface
- <sys/stat.h> : file status
- <sys/times.h> : process times
- <sys/types.h> primitive system datatypes
- <sys/un.h> : UNIX domain socket definition
- <sys/utsname.h> : system name
- <sys/wait.h> process control

Exercise Setup

- You are given a task to keep track of number of queries running in database management system. Below information regarding queries need to be kept in C structure.

```
struct query {
int query_id;           // unique id
char* query_text;  // actual query text
int status;     // submitted, running, finished
int time_elapsed;   // time in microsec it took to run the query
}
```

Exercise Problem 1

- Create an array of structure for struct query defined previously using dynamic memory allocation and to provide the following functions. For each of functionality you can use user menu selection
  - New query is submitted by a SQL developer i.e. new array entry is added with status as submitted
  - Query has started running so update to running
  - Query is running so periodically need to update time_elapsed
  - Query has finished running so update the status as finished and remove the entry from array

Exercise Problem 2

- Functionality of Problem 2 is same as Problem1
  - New query is submitted by a SQL developer i.e. new array entry is added with status as submitted
  - Query has started running so update to running
  - Query is running so periodically need to update time_elapsed
  - Query has finished running so update the status as finished and remove the entry from array
- But Problem 2 should be implemented using linked list instead of array

- Complete the code in C_revision\excercise2.c