S&T2024

Computer Programming

(Part 2 – Advanced C Programming Language)

Chapter 0

Lecturer
A/Prof Tay Seng Chuan

E-mail: pvotaysc@nus.edu.sg

Office of the Provost
National University of Singapore

Ground Rules

- Switch off your handphone and pager
- No talking while lecture is going on
- No gossiping while the lecture is going on
- Raise your hand if you have question to ask
- Be on time for lecture
- Be on time to come back from the recess break to continue the lecture
- Bring your printed lecture notes to lecture or use a laptop for the e-copy. Do not use handphone to read the pdf file.
- Do not open any app, except this pdf file and an editing tool (to take notes).

Chapter 0

Elementary Data
Structure
with struct

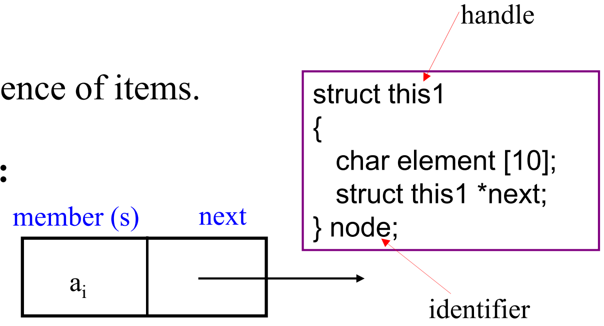
Data Structures

Data structure is an organizational scheme, such as a structure, array, or pointer that can be applied to data to facilitate interpreting the data or performing operations on it.

Linked List

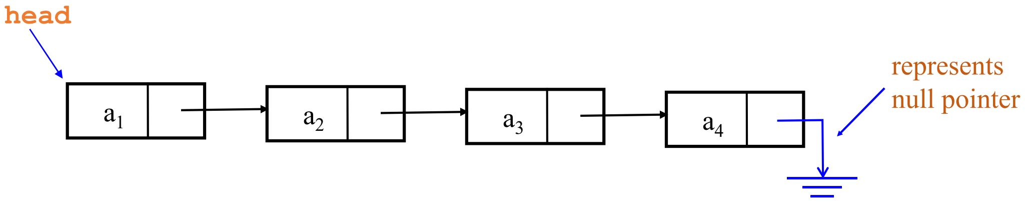
A linked list is a sequence of items.

A node in linked list:



```
struct this1
{
   char element [10];
   struct this1 *next;
} node;
```

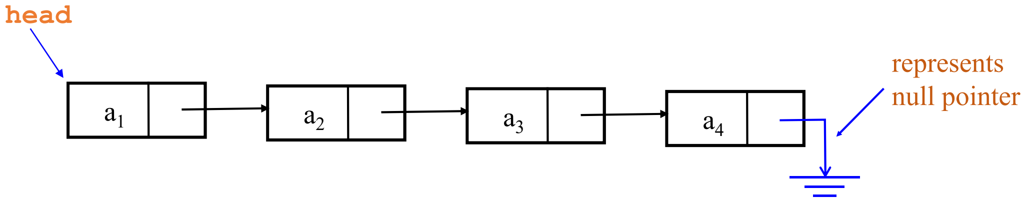
The linked list:



head is a pointer, and null is a pointer that has the address 0.

One-Way Linked List Representation

- O(n) as opposed to an array O(1) access time
- In a linked list we have to start at the first position. It is done by a pointer (usually named as **head**.)



- A linked list will have to be terminated by a null pointer.

What is **Word Alignment** in memory allocation in general?

Word alignment refers to arranging data in memory so that variables start at memory addresses that are multiples of the word size.

On a 32-bit system: A word is 4 bytes. Word-aligned data means that

- int (4 bytes) should start at addresses divisible by 4 (e.g., 0, 4, 8, 12…)
- double (8 bytes) might require 8-byte alignment (even on 32-bit systems, depending on compiler).

Why Alignment Matters:

- Speed: Aligned data can be accessed faster by the CPU. **Why?**
- Hardware requirements: Some CPUs can't read misaligned data without errors or extra cycles.
- Padding: The compiler automatically adds padding bytes to align members inside structs.

Consider 32-bit setting:

```
typedef  struct animal
{
   char name [10];
   struct animal *next;
} node;
```

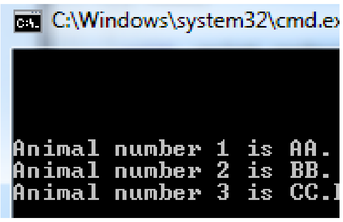
node will be replaced by the structure

Address (pointer can have only 32 bits):

$$\frac{32 \text{ bit}}{8 \text{ bit}} = 4 \text{ bytes}$$

Pointer should start at the byte address of multiple of 4.

Number of bytes for this struct ?



```
Animal number 1 is AA.
Animal number 2 is BB.
Animal number 3 is CC.I
```

link1.c

Consider 64-bit setting:

```
typedef  struct animal
{
   char name [10];
   struct animal *next;
} node;
```

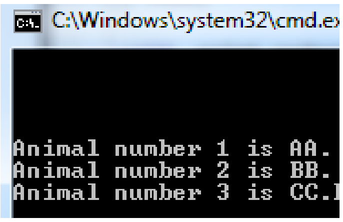
node will be replaced by the structure

Address (pointer can have 64 bits in the 64-bit setting):

$$\frac{64 \text{ bit}}{8 \text{ bit}} = 8 \text{ bytes}$$

Pointer in the 64-bit setting needs 8 bytes thus should start at the byte address of multiple of 8.

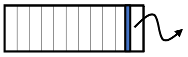
Number of bytes for the struct ?

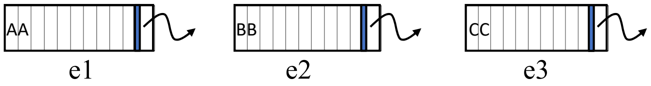


```
Animal number 1 is AA.
Animal number 2 is BB.
Animal number 3 is CC.I
```

link1.c

```
// link1.c - linked list
#include <stdio.h>
/* declare a self-referential structure */

typedef  struct animal
{
   char name [10];
   struct animal *next;
} node;

void  print_nodes(const node *ptr);

main()
{   /* define three node variables and
       one pointer to node */

    node e1, e2, e3, *start;

    /* store nodes' names */
    strcpy( e1.name, "AA" );
    strcpy( e2.name, "BB" );
    strcpy( e3.name, "CC" );
    /* link nodes */

    e1.next = &e2;
    e2.next = &e3;
    e3.next = NULL;
```





```
    /* start contains the address of the first node */
    start = &e1;
    print_nodes (start);
    return 0;
}

void  print_nodes (const node *ptr)
{
    int count = 1;
    printf( "\n\n\n" );
    while ( ptr !=  NULL )
    {
        printf( "\nAnimal number %d is %s.",
                      count++,   ptr -> name );
        ptr = ptr -> next;
    }
}
```

You have to use **#include <stdlib.h> for dynamic memory allocation.**

C library function – malloc ()

The C library function **void *malloc(size_t size)** allocates the requested memory and returns a pointer to it

Parameters

size - This is the size of the memory block, in bytes.

Return Value

This function returns a pointer to the allocated memory, or NULL if the request fails.

C library function - free()

The C library function **void free(void *ptr)** deallocates (returns) the memory previously allocated by a call to calloc, **malloc**, or realloc.

Parameters

ptr - This is the pointer to a memory block previously allocated with malloc, calloc or realloc to be deallocated. If a null pointer is passed as argument, no action occurs.

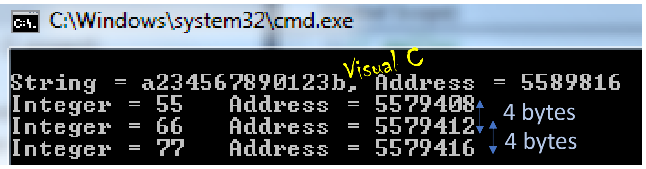
Return Value

This function does not return any value.

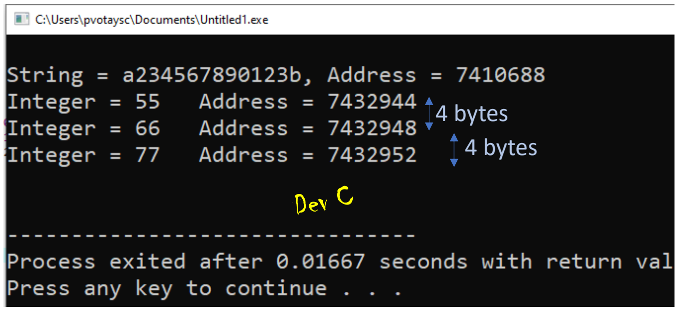
http://www.tutorialspoint.com/c_standard_library/c_function_malloc.htm

Example

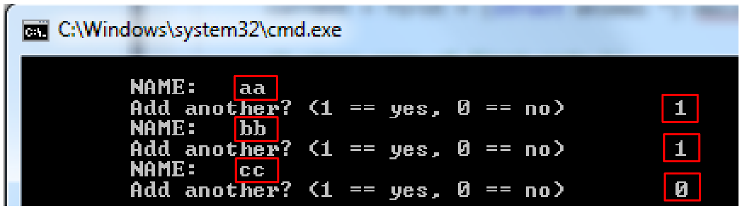
```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *this1;
    int  *this2;
    int size;

    this1 = (char *) malloc(15);
    strcpy (this1, "a234567890123b");
    printf("\nString = %s, Address = %u\n", this1, this1);

    size = sizeof (int);

    this2 = (int *) malloc(size*3);
    this2[0]=55;  this2[1]=66;  this2[2]=77;

    printf("Integer = %d   Address = %u\n", this2[0], &this2[0]);
    printf("Integer = %d   Address = %u\n", this2[1], &this2[1]);
    printf("Integer = %d   Address = %u\n\n", this2[2], &this2[2]);

   free (this1);
   free (this2);

   return 0;
}  // 4-byte address for Visual C; 8-byte address for Dev C; integer takes 4 bytes
```


C:\Windows\system32\cmd.exe

```
String = a234567890123b, Address = 5589816
Integer = 55    Address = 5579408
Integer = 66    Address = 5579412
Integer = 77    Address = 5579416
```


C:\Users\pvotaysc\Documents\Untitled1.exe

```
String = a234567890123b, Address = 7410688
Integer = 55    Address = 7432944
Integer = 66    Address = 7432948
Integer = 77    Address = 7432952


--------------------------------
Process exited after 0.01667 seconds with return val
Press any key to continue . . .
```



```
NAME:    aa
Add another? (1 == yes, 0 == no)         1
NAME:    bb
Add another? (1 == yes, 0 == no)         1
NAME:    cc
Add another? (1 == yes, 0 == no)         0
```

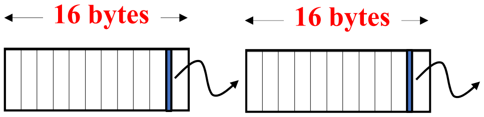
link2.c

link2.c - linked list

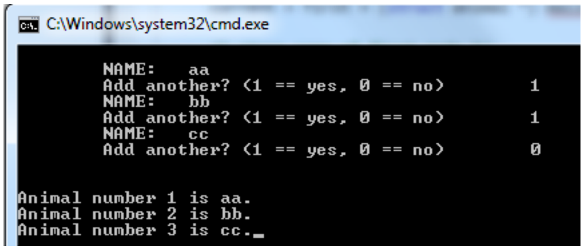
This struct will need 24 bytes on 64-bit setting.

```
#include <stdio.h>

/* declare a self-referential structure */

typedef  struct animal
{
   char   name [10];
   struct animal  *next;
} node;   // (12+4) = 16 bytes if 1 word = 4 bytes
```

node will be replaced by the structure



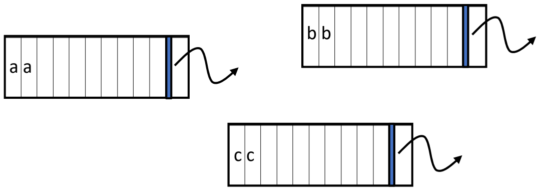
```
void print_nodes (const node *ptr );
node *get_nodes( void ), *start;

main()
{
    start = get_nodes ();
    print_nodes (start);
    return 0;
}
```



```
        NAME:    aa
        Add another? (1 == yes, 0 == no)         1
        NAME:    bb
        Add another? (1 == yes, 0 == no)         1
        NAME:    cc
        Add another? (1 == yes, 0 == no)         0

Animal number 1 is aa.
Animal number 2 is bb.
Animal number 3 is cc._
```

P.T.O.

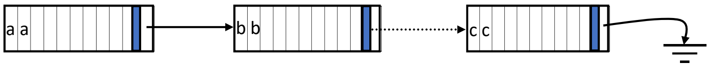
```
/*   get_nodes allocates run-time storage for nodes. It builds the linked list and stores user-supplied
     names in the name fields of the nodes. It returns a pointer to the first such node. */

node *get_nodes( void )
{
    node  *current, *first;
    int response;

    /* allocate first node */
    first = current =
          (struct animal *) malloc( sizeof (node) );

    /* store name of first node */
    printf( "\n\tNAME:\t" );
    scanf( "%s", current -> name );

    /* prompt user about another node */
    printf( "\tAdd another? (1 = yes, 0 = no)\t" );
    scanf( "%d", &response );
```



```
   /* Add nodes to list until user signals halt. */

    while ( response )
    {
        /* try to allocate another node node */
        if ( ( current -> next =
            (struct animal *) malloc( sizeof ( node ) ) ) == NULL )
        {
            printf( "Out of memory\nCan't add more nodes\n" );
            return first;
         }
         current = current -> next;

         /* store name of next node */
         printf( "\tNAME:\t" );
         scanf( "%s", current -> name );
         /* prompt user about another node */
         printf( "\tAdd another? (1 = yes, 0 = no)\t" );
         scanf( "%d", &response );
    }

    /* set link field in last node to NULL */
    current -> next = NULL;
    return first;
}
```

P.T.O.

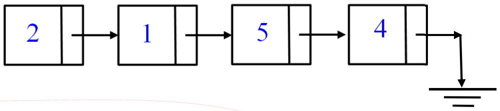
```
void  print_nodes( const node *ptr )
{
    int count = 1;
    printf( "\n" );
    while ( ptr !=  NULL )
    {
        printf( "\nAnimal number %d is %s.",
                           count++, ptr -> name );
        ptr = ptr -> next;
    }
}

node *start;

 main()
 {
     start = get_nodes();
     print_nodes( start );
     return 0;
 }
```



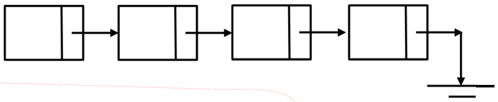
Example: Compute the Average on List

(head is a pointer to structure)

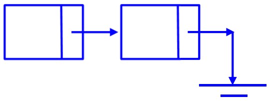
```
double average (listptr head)
{
  double sum = 0;
  int n = 0;

  if (head == NULL)
  {
    printf (" \n empty list ");
    exit (1);
  }

  do
  {
    n++;
    sum += head->value; // to access to the field of pointer to struct
    head = head->next;  //        head.value is wrong
  }
  while (head != NULL);

  return sum/n;
}
```



Example: Concatenate 2 lists



```
listptr concatlists (listptr first, listptr second)
{
  listptr temp;

  if (first == NULL)
    return second;

  if (second != NULL)
  {
    temp = first;
    while (temp->next != NULL) temp = temp->next;

    temp->next = second;
  }

  return first;
}
```

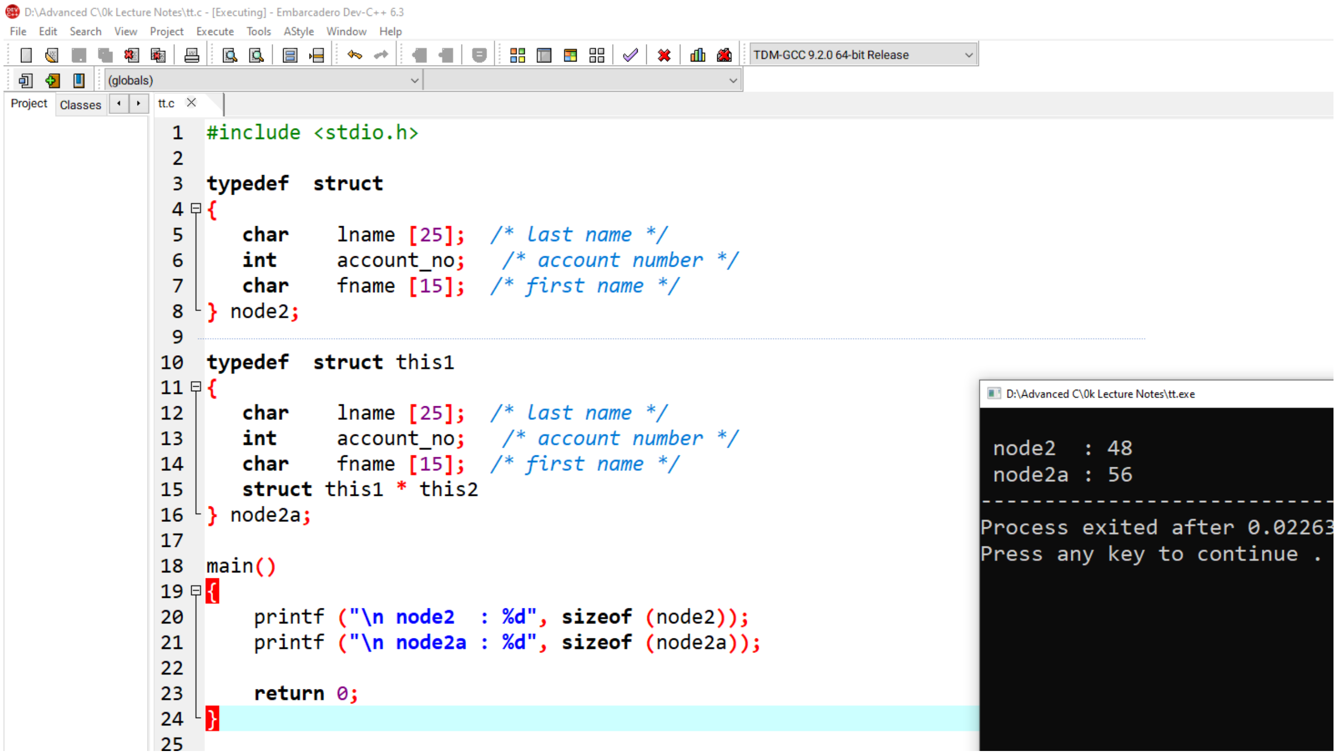


What are the characteristics of a queue?

First in First Out (First Come First Serve)

```
typedef  struct
{
  char    lname[ 25 ];  /* last name */
  int      account_no;   /* account number */
  char    fname[ 15 ];  /* first name */
} node2;
```

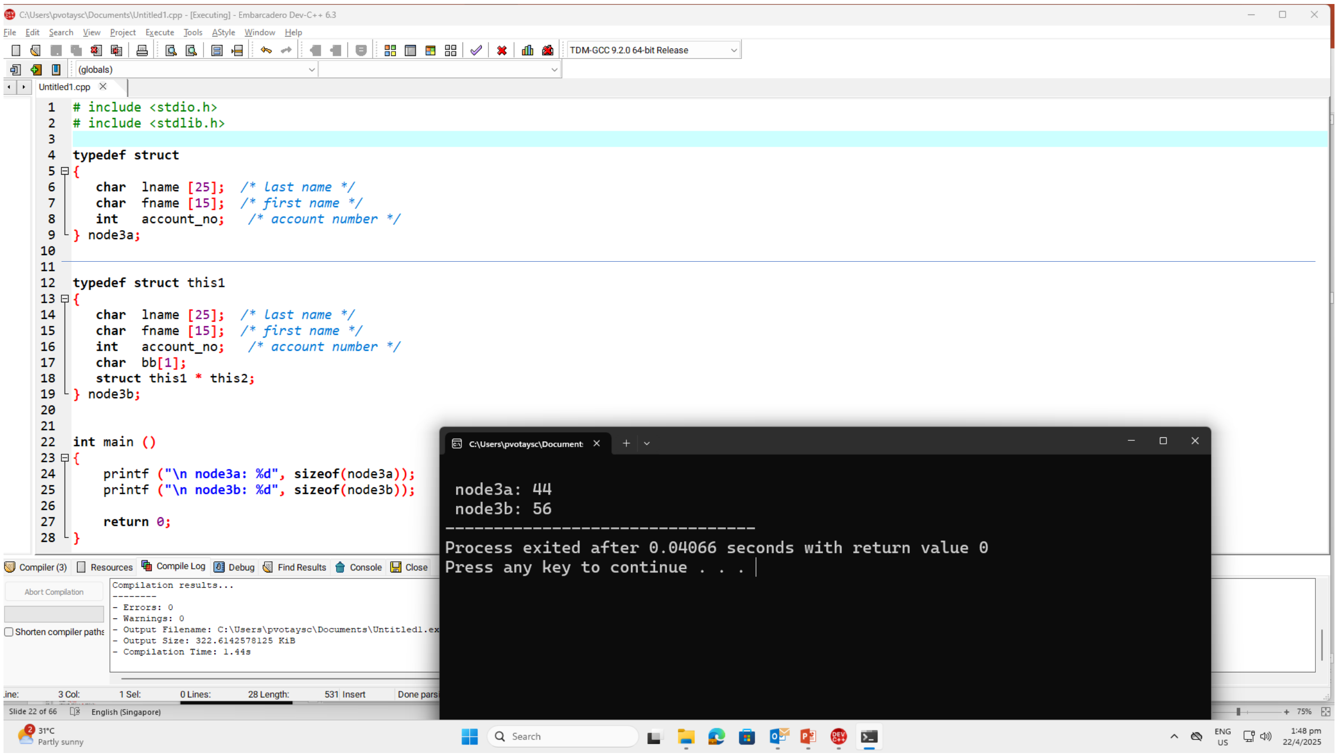
```
typedef  struct
{
  char    lname[ 25 ];  /* last name */
  char    fname[ 15 ];  /* first name */
  int      account_no;   /* account number */
} node3;
```


D:\Advanced C\0k Lecture Notes\tt.c - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 9.2.0 64-bit Release
(globals)
Project Classes
tt.c

```c
#include <stdio.h>

typedef  struct
{
   char    lname [25];  /* last name */
   int     account_no;   /* account number */
   char    fname [15];  /* first name */
} node2;

typedef  struct this1
{
   char    lname [25];  /* last name */
   int     account_no;   /* account number */
   char    fname [15];  /* first name */
   struct this1 * this2
} node2a;

main()
{
    printf ("\n node2  : %d", sizeof (node2));
    printf ("\n node2a : %d", sizeof (node2a));

    return 0;
}
```

D:\Advanced C\0k Lecture Notes\tt.exe

```
 node2  : 48
 node2a : 56
--------------------------------
Process exited after 0.02263
Press any key to continue .
```


C:\Users\pvotaysc\Documents\Untitled1.cpp - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 9.2.0 64-bit Release
(globals)
Untitled1.cpp

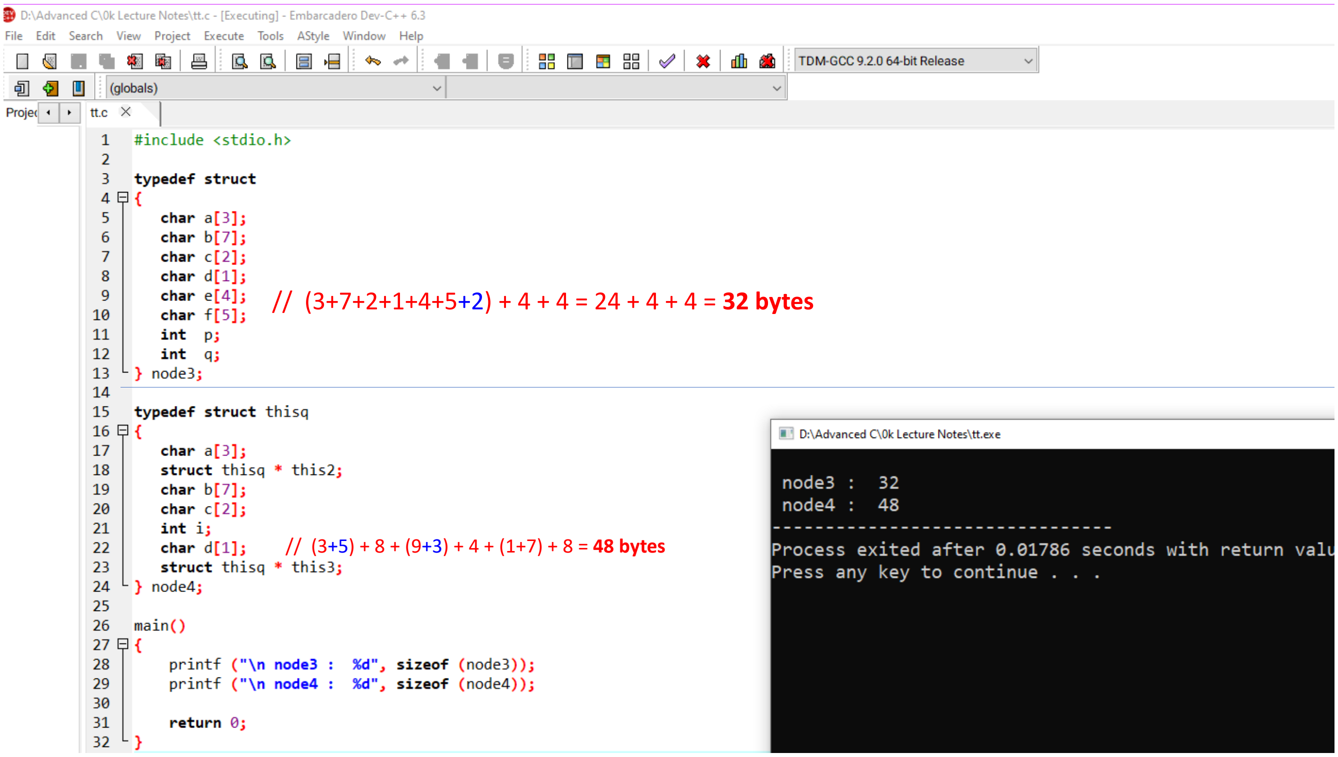
```c
# include <stdio.h>
# include <stdlib.h>

typedef struct
{
   char  lname [25];  /* last name */
   char  fname [15];  /* first name */
   int   account_no;   /* account number */
} node3a;


typedef struct this1
{
   char  lname [25];  /* last name */
   char  fname [15];  /* first name */
   int   account_no;   /* account number */
   char  bb[1];
   struct this1 * this2;
} node3b;


int main ()
{
    printf ("\n node3a: %d", sizeof(node3a));
    printf ("\n node3b: %d", sizeof(node3b));

    return 0;
}
```

Compiler (3)
Resources
Compile Log
Debug
Find Results
Console
Close
Abort Compilation
□ Shorten compiler paths

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\pvotaysc\Documents\Untitled1.ex
- Output Size: 322.6142578125 KiB
- Compilation Time: 1.44s
```

Line: 3 Col: 1 Sel: 0 Lines: 28 Length: 531 Insert Done parsi
Slide 22 of 66
English (Singapore)
31°C
Partly sunny
Search
ENG US
1:48 pm
22/4/2025
C:\Users\pvotaysc\Document

```
 node3a: 44
 node3b: 56
--------------------------------
Process exited after 0.04066 seconds with return value 0
Press any key to continue . . .
```


D:\Advanced C\0k Lecture Notes\tt.c - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 9.2.0 64-bit Release
(globals)
Projec
tt.c
D:\Advanced C\0k Lecture Notes\tt.exe

```
#include <stdio.h>

typedef struct
{
    char a[3];
    char b[7];
    char c[2];
    char d[1];
    char e[4];   // (3+7+2+1+4+5+2) + 4 + 4 = 24 + 4 + 4 = 32 bytes
    char f[5];
    int  p;
    int  q;
} node3;

typedef struct thisq
{
    char a[3];
    struct thisq * this2;
    char b[7];
    char c[2];
    int i;
    char d[1];     // (3+5) + 8 + (9+3) + 4 + (1+7) + 8 = 48 bytes
    struct thisq * this3;
} node4;

main()
{
     printf ("\n node3 :  %d", sizeof (node3));
     printf ("\n node4 :  %d", sizeof (node4));

     return 0;
}
```

```
 node3 :  32
 node4 :  48
--------------------------------
Process exited after 0.01786 seconds with return valu
Press any key to continue . . .
```

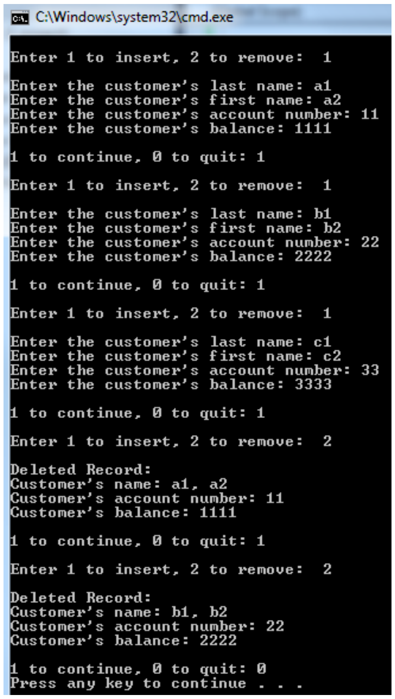
queue1.c

Array of pointers to structure

```
typedef  struct
{
   char    lname[ 25 ];  /* last name */
   char    fname[ 15 ];  /* first name */
   int     account_no;   /* account number */
   long int   balance;   /* balance */
} node;
```

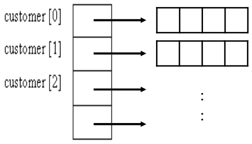
// Visual C and Dev C do not differentiate int and long int
// Each of int or long int will take 4 bytes

sizeof (node) = (25+15+0)+ 4 + 4 = **48 bytes**



```
Enter 1 to insert, 2 to remove:  1

Enter the customer's last name: a1
Enter the customer's first name: a2
Enter the customer's account number: 11
Enter the customer's balance: 1111

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove:  1

Enter the customer's last name: b1
Enter the customer's first name: b2
Enter the customer's account number: 22
Enter the customer's balance: 2222

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove:  1

Enter the customer's last name: c1
Enter the customer's first name: c2
Enter the customer's account number: 33
Enter the customer's balance: 3333

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove:  2

Deleted Record:
Customer's name: a1, a2
Customer's account number: 11
Customer's balance: 1111

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove:  2

Deleted Record:
Customer's name: b1, b2
Customer's account number: 22
Customer's balance: 2222

1 to continue, 0 to quit: 0
Press any key to continue . . .
```

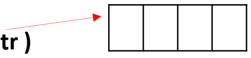
```
// queue1.c  - FIFO Queue
// Add node to rear, and delete fron front.     Array is used.
#include <stdio.h>
#define  SIZE  100

typedef  struct
{
   char    lname[ 25 ];  /* last name */
   char    fname[ 15 ];  /* first name */
   int     account_no;   /* account number */
   long int   balance;   /* balance */
} node;

node  *customers[SIZE];  //array of 100 pointers
int   front = 0, rear = 0;   /* exit and entry positions in queue */
int   count = 0;             /* count of items in queue */
node *insert(node), *delete();
void  get_data(node *), put_data(const node *);

main()
{
    int ans, flag;
    node this1, *ptr;
```



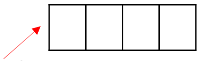
P.T.O.

```
do {    /* do queue operations until user signals halt */
     do {
          printf( "\nEnter 1 to insert, 2 to remove:  " );
          scanf( "%d", &ans );
          printf( "\n" );
          switch (ans)
          {
            case 1:
             get_data( &this1);
             if ( insert(this1) == NULL )  printf( "\nQUEUE FULL\n\n" );
             break;
           case 2:
             if ( ( ptr = delete() ) != NULL )
               put_data( ptr );
             else
               printf( "\n\nQUEUE EMPTY\n\n" );
             break;
           default:
             printf( "\nIllegal response\n" );
             break;
          }
     } while ( ans != 1 && ans != 2 );
     printf( "1 to continue, 0 to quit: " );
     scanf( "%d", &flag );
  } while ( flag );
  return 0;
}
```

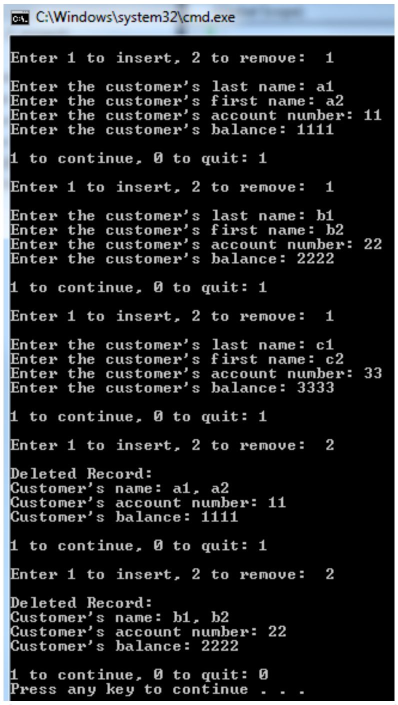
P.T.O.



```
void  get_data(node *ptr )
{
    printf( "Enter the customer's last name: " );
    scanf( "%s", ptr -> lname );
    printf( "Enter the customer's first name: " );
    scanf( "%s", ptr -> fname );
    printf( "Enter the customer's account number: " );
    scanf( "%d", &( ptr -> account_no ) );
    printf( "Enter the customer's balance: " );
    scanf( "%ld", &( ptr -> balance ) );
    printf( "\n" );
}
```



```
void put_data( const node *ptr )
{
    printf ("Deleted Record:");
    printf( "\nCustomer's name: %s, %s\n",
                    ptr -> lname,  ptr -> fname  );
    printf( "Customer's account number: %d\n",
                    ptr -> account_no );
    printf( "Customer's balance: %ld  \n\n", ptr -> balance );
}
```



C:\Windows\system32\cmd.exe

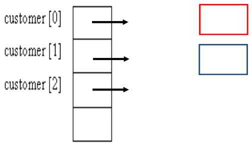
```
Enter 1 to insert, 2 to remove:  1

Enter the customer's last name: a1
Enter the customer's first name: a2
Enter the customer's account number: 11
Enter the customer's balance: 1111

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove:  1

Enter the customer's last name: b1
Enter the customer's first name: b2
Enter the customer's account number: 22
Enter the customer's balance: 2222

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove:  1

Enter the customer's last name: c1
Enter the customer's first name: c2
Enter the customer's account number: 33
Enter the customer's balance: 3333

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove:  2

Deleted Record:
Customer's name: a1, a2
Customer's account number: 11
Customer's balance: 1111

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove:  2

Deleted Record:
Customer's name: b1, b2
Customer's account number: 22
Customer's balance: 2222

1 to continue, 0 to quit: 0
Press any key to continue . . .
```

P.T.O.

```
node *insert(node this1)
{
    node *ptr;
    if ( count  >=  SIZE ) return NULL;    /* queue full? */
                                              48
    ptr = (node *) malloc ( sizeof (node) );
    *ptr = this1;                      /* store data */
    customers[ rear ] = ptr;    /* add to queue */
    rear = ++rear % SIZE;    /* update rear */
    ++count;                           /* update count */
    return ptr;
}
```

```
node *delete( void )
{
    static node this1;
    if ( count == 0 ) return NULL;       /* empty queue? */
    this1 = *customers[front];
    free( customers[front] );    /* collect garbage */
    front = ++front % SIZE;    /* update front */
    --count;
    return &this1;
}
```

```
#include <stdio.h>     // queue1.c  - FIFO Queue

#define  SIZE  100
typedef  struct
{
   char    lname[ 25 ];  /* last name */
   char    fname[ 15 ];  /* first name */
   int     account_no;   /* account number */
   long int   balance;   /* balance */
} node;
node  *customers[SIZE];
int    front = 0, rear = 0;   /* exit and entry positions in queue */
int    count = 0;             /* count of items in queue */
node *insert(node), *delete();
void  get_data(node *), put_data(const node *);

main()
{
    int ans, flag;
    node this1,                  *ptr;
    clrscr();
    do {
         do {
              printf( "\nEnter 1 to insert, 2 to remove:  " );
              scanf( "%d", &ans );
              printf( "\n" );
              switch (ans)  {
                 case 1:
                    get_data( &this1);
                    if ( insert(this1) == NULL ) printf( "\nQUEUE FULL\n\n" );
                    break;
                 case 2:
                    if ( ( ptr = delete() ) != NULL ) put_data( ptr );
           else  printf( "\n\nQUEUE EMPTY\n\n" );
                    break;
                 default:
                    printf( "\nIllegal response\n" );
                    break;
              }
         } while ( ans != 1 && ans != 2 );
         printf( "1 to continue, 0 to quit: " );  scanf( "%d", &flag );
    } while ( flag );
    return 0;
}
```

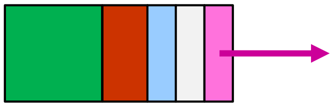


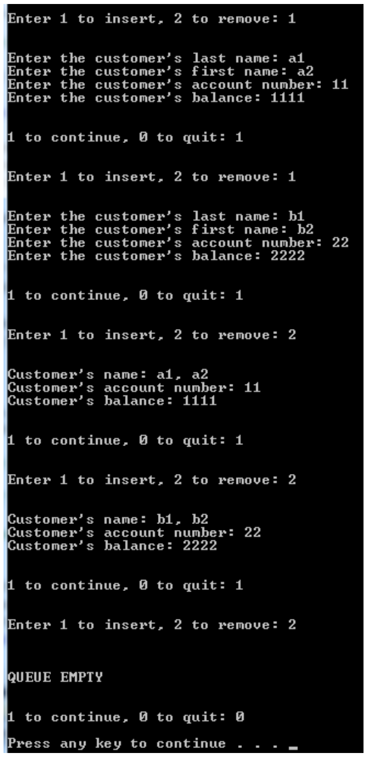
```
void  get_data(node *ptr )
{
    printf( "Enter the customer's last name: " );
    scanf( "%s", ptr -> lname );
    printf( "Enter the customer's first name: " );
    scanf( "%s", ptr -> fname );
    printf( "Enter the customer's account number: " );
    scanf( "%d", &( ptr -> account_no ) );
    printf( "Enter the customer's balance: " );
    scanf( "%ld", &( ptr -> balance ) );
    printf( "\n" );
}
void put_data( const node *ptr )
{
    printf ("Deleted Record:");
    printf( "\nCustomer's name: %s, %s\n",
                 ptr -> lname, ptr -> fname  );
    printf( "Customer's account number: %d\n",
                                 ptr -> account_no );
    printf( "Customer's balance: %ld  \n\n", ptr -> balance );
}
node *insert(node this1)
{
    node *ptr;
    if ( count  >=  SIZE ) /* queue full? */
       return NULL;
    ptr = (node *) malloc( sizeof (node) );
    *ptr = this1;            /* store data */
    customers[ rear ] = ptr;  /* add to queue */
    rear = ++rear % SIZE;     /* update rear */
    ++count;                  /* update count */
    return ptr;
}
node *delete( void )
{
    static node this1;
    if ( count == 0 )   /* empty queue? */
      return NULL;
    this1 = *customers[front];
    free( customers[front] );  /* collect garbage */
    front = ++front % SIZE;    /* update front */
    --count;
    return &this1;
}
```

queue2.c

Linked List

```
typedef  struct customer
{
  char    lname[ 25 ];     /* last name */
  char    fname[ 15 ];     /* first name */
  int     account_no;      /* account number */
  long int   balance;      /* balance */
  struct  customer * succ; /* successor on the queue */
} node;
```



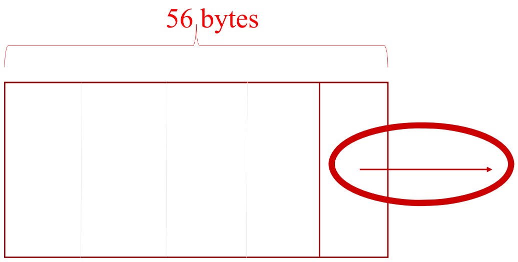


```
Enter 1 to insert, 2 to remove: 1

Enter the customer's last name: a1
Enter the customer's first name: a2
Enter the customer's account number: 11
Enter the customer's balance: 1111

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove: 1

Enter the customer's last name: b1
Enter the customer's first name: b2
Enter the customer's account number: 22
Enter the customer's balance: 2222

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove: 2

Customer's name: a1, a2
Customer's account number: 11
Customer's balance: 1111

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove: 2

Customer's name: b1, b2
Customer's account number: 22
Customer's balance: 2222

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove: 2


QUEUE EMPTY

1 to continue, 0 to quit: 0
Press any key to continue . . . _
```

What if

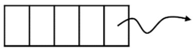
```
typedef  struct customer
{
  char    lname[ 25 ];     /* last name */
  char    fname[ 15 ];     /* first name */
  int     account_no;      /* account number */
  struct  customer * succ; /* successor on the queue */
} node;
```

```
// queue2.c - FIFO Queue

// This program add new node to the rear of the queue, and
// delete node from the front of the queue
// pointer is used

#include <stdio.h>
#define  SIZE  100

typedef  struct customer
{
  char   lname[ 25 ];    /* last name */
  char   fname[ 15 ];    /* first name */
  int     account_no;     /* account number */
  long int   balance;     /* balance */
  struct  customer * succ; /* successor on the queue */
} node;

node *front, *rear;   /* exit entry positions in queue */
int  count = 0;          /* count of items in queue */
void  get_data (node *),  put_data(const node *);
node *insert (node), *delete();
```



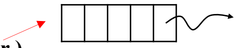
P.T.O.

```
main ()
{
    int  ans, flag;
    node this1, *ptr;
```

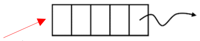


```
do {       /* do queue operations until user signals halt */
      do {
          printf( "\nEnter 1 to insert, 2 to remove: " );
          scanf( "%d", &ans );
          printf( "\n" );
          switch ( ans )
          {
             case 1: /* get a node and add to queue */
                get_data( &this1);
                if ( insert(this1) == NULL )
                     printf( "\n\nQUEUE FULL\n\n" );
                break;
             case 2: /* delete a node from queue and print */
                if ( ( ptr = delete() ) != NULL )
                     put_data( ptr );
                else
                     printf( "\n\nQUEUE EMPTY\n\n" );
                break;
             default:
                printf( "\nIllegal response\n" );
                break;
          }
      } while ( ans != 1 && ans != 2 );
      printf( "\n1 to continue, 0 to quit: " );
      scanf( "%d", &flag );
      printf( "\n" );
  } while ( flag );
  return 0;
}
```

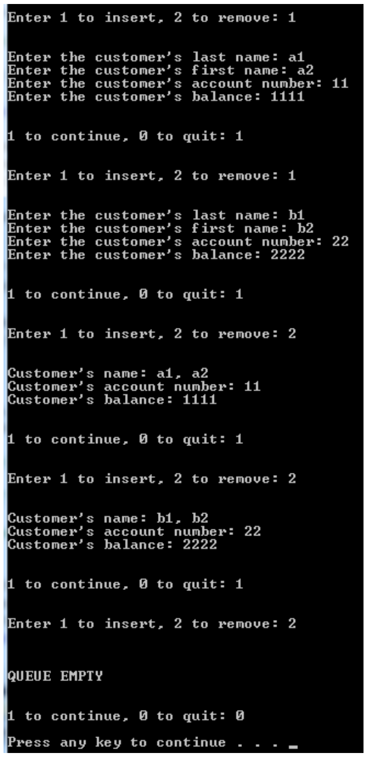
P.T.O.



```
void  get_data(node *ptr )
{
    printf( "\nEnter the customer's last name: " );
    scanf( "%s", ptr -> lname );
    printf( "Enter the customer's first name: " );
    scanf( "%s", ptr -> fname );
    printf( "Enter the customer's account number: " );
    scanf( "%d", &( ptr -> account_no ) );
    printf( "Enter the customer's balance: " );
    scanf( "%ld", &( ptr -> balance ) );
    printf( "\n" );
}
```



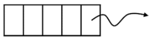
```
void put_data( const node *ptr )
{
    printf( "\nCustomer's name: %s, %s\n", ptr -> lname,
         ptr -> fname  );
    printf( "Customer's account number: %d\n",
         ptr -> account_no );
    printf( "Customer's balance: %ld \n\n", ptr -> balance );
}
```

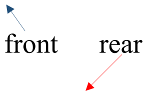


```
Enter 1 to insert, 2 to remove: 1

Enter the customer's last name: a1
Enter the customer's first name: a2
Enter the customer's account number: 11
Enter the customer's balance: 1111

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove: 1

Enter the customer's last name: b1
Enter the customer's first name: b2
Enter the customer's account number: 22
Enter the customer's balance: 2222

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove: 2

Customer's name: a1, a2
Customer's account number: 11
Customer's balance: 1111

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove: 2

Customer's name: b1, b2
Customer's account number: 22
Customer's balance: 2222

1 to continue, 0 to quit: 1

Enter 1 to insert, 2 to remove: 2


QUEUE EMPTY

1 to continue, 0 to quit: 0

Press any key to continue . . . _
```

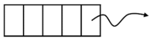
P.T.O.

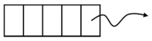
```
/*   If the queue is full, insert returns NULL. Otherwise, insert
     allocates storage for a node, copies the data passed into
     the allocated storage, adds the node to the rear (last node in
     the linked list), updates rear, NULLs the link field of the new
     node, updates count, and returns the address of the node
     added. */
```





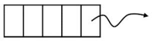
```
node *insert(node this1)
{
    node *ptr;
    // if ( count  >=  SIZE ) return NULL;     /* queue full? */

    ptr = (node *) malloc( sizeof (node) ); /* new customer (node) */
    *ptr = this1;     /* store data */

    if ( count == 0 ) /* empty queue? */
      front = ptr;      /* front points to first node in list */
    else
      rear -> succ = ptr; /* if queue not empty, add at end */

    rear = ptr;         /* update rear */
    ptr -> succ = NULL; /* null the last succ field */
    ++count;            /* update count */
    return ptr;
}
```

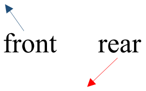




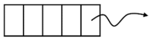
P.T.O.

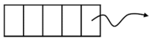
```
/*  If the queue is empty, delete returns NULL. Otherwise, delete
    copies the node at the front (first node in the linked list)
    to permanent storage, updates front, frees the node, updates
    count, and returns the address of the node.              */

node *delete( void )
{
    static node this1;
    node *next;
    if (count == 0) return NULL;    /* empty queue? */

    this1 = *front;    /* copy node at front */
    next = front;      /* save front node's address for freeing */
    front = front -> succ;  /* remove front node */

    free( next );  /* deallocate storage */
    --count;       /* update count */

    return &this1;
}
```

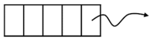


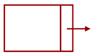


```
// queue2.c - FIFO Queue
// This program add new node to the rear of the queue, and delete node
// from the front of the queue,  pointer is used
#include <stdio.h>
#define  SIZE  100

typedef  struct customer
{
   char    lname[ 25 ];     /* last name */
   char    fname[ 15 ];     /* first name */
   int     account_no;      /* account number */
   long int   balance;      /* balance */
   struct  customer * succ; /* successor on the queue */
} node;
node *front, *rear;    /* exit entry positions in queue */
int  count = 0;        /* count of items in queue */
void  get_data (node *), put_data(const node *);
node *insert (node), *delete();
main()
{
   int  ans, flag;
   node this1,                    *ptr;
   // clrscr();
   do {
    do {
         printf( "\nEnter 1 to insert, 2 to remove: " );
        scanf( "%d", &ans ); printf( "\n" );
        switch ( ans )
        {  case 1: /* get a node and add to queue */
              get_data( &this1);
              if ( insert(this1) == NULL )  printf( "\n\nQUEUE FULL\n\n" );
              break;
           case 2: /* delete a node from queue and print */
              if ( ( ptr = delete() ) != NULL ) put_data( ptr );
              else printf( "\n\nQUEUE EMPTY\n\n" );
              break;
           default:
              printf( "\nIllegal response\n" );
              break;
         }
     } while ( ans != 1 && ans != 2 )
      printf( "\n1 to continue, 0 to quit: " ); scanf( "%d", &flag ); printf( "\n" );
   } while ( flag );
   return 0;
}
```

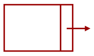


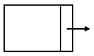




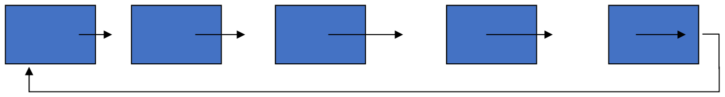


```
void  get_data(node *ptr )
{
    printf( "\nEnter the customer's last name: " );
    scanf( "%s", ptr -> lname );
    printf( "Enter the customer's first name: " );
    scanf( "%s", ptr -> fname );
    printf( "Enter the customer's account number: " );
    scanf( "%d", &( ptr -> account_no ) );
    printf( "Enter the customer's balance: " );
    scanf( "%ld", &( ptr -> balance ) );
    printf( "\n" );
}
void put_data( const node *ptr )
{
    printf( "\nCustomer's name: %s, %s\n", ptr -> lname, ptr -> fname );
    printf( "Customer's account number: %d\n", ptr -> account_no );
    printf( "Customer's balance: %ld \n\n", ptr -> balance );
}
node *insert(node this1)
{
    node *ptr;
    // if ( count  >=  SIZE ) return NULL;     /* queue full? */
    ptr = (node *) malloc( sizeof (node) ); /* new customer (node) */
    *ptr = this1;     /* store data */
    if ( count == 0 ) /* empty queue? */
      front = ptr;    /* front points to first node in list */
    else
      rear -> succ = ptr; /* if queue not empty, add at end */
    rear = ptr;         /* update rear */
    ptr -> succ = NULL; /* NULL last succ field */
    ++count;            /* update count */
    return ptr;
}
node *delete( void )
{
    static node this1;
    node *next;
    if (count == 0) return NULL;     /* empty queue? */
    this1 = *front;     /* copy node at front */
    next = front;       /* save front node's address for freeing */
    front = front -> succ; /* remove front node */
    free( next );  /* deallocate storage */
    --count;       /* update count */
    return &this1;
}
```



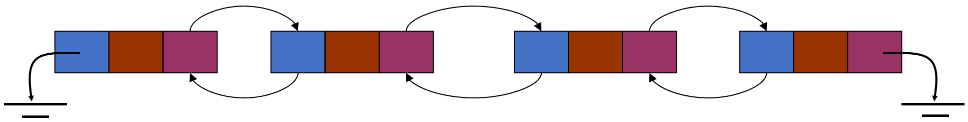


Circular Linked List



Given a pointer to an arbitrary node on a Circular Linked List, we can follow links from a node to access any other node.

Two Way Linked List



Point to both their left and right neighbours, you can follow links in either direction to access other nodes.

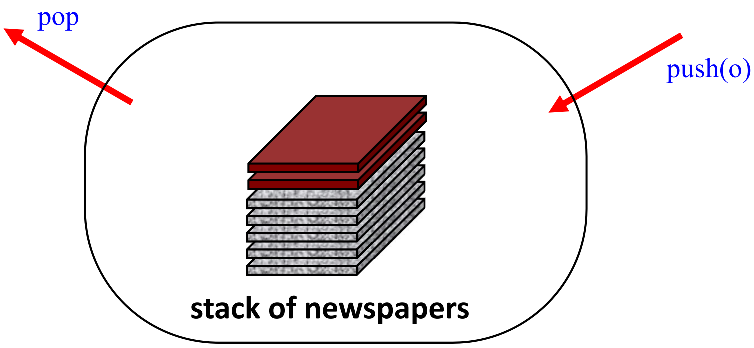
```
typedef  struct customer
{
   char    lname[ 25 ];     /* last name */
   char    fname[ 15 ];     /* first name */
   int     account_no;      /* account number */
   long int   balance;      /* balance */
   struct  customer * succ; /* successor on the queue */
} node;
```

```
typedef  struct customer
{
   char    lname[ 25 ];     /* last name */
   char    fname[ 15 ];     /* first name */
   int     account_no;      /* account number */
   long int   balance;      /* balance */
   struct  customer * right;   /* point to right*/
   struct  customer * left;      /* point to lrft*/
} node2;
```

Stack

What is a Stack?

- Stacks can be implemented efficiently and are very useful in computing.
- Stacks exhibit the LIFO behaviour.



Applications

Many application areas use stacks:

- line editing
- bracket matching
- postfix calculation
- function call stack

Line Editing

A line editor would place the characters read into a buffer but may use a backspace symbol (denoted by ←) to do error correction.

Refined Task
- read in a line
- correct the errors via backspace
- print the corrected line in reverse

Example:

Input : `abc_defg~~h~~←2klp~~qr~~←←wxyz`

Corrected Input : `abc_defg2klpwxyz`

Reversed Output : `zyxwplk2gfed_cba`

Informal Procedure

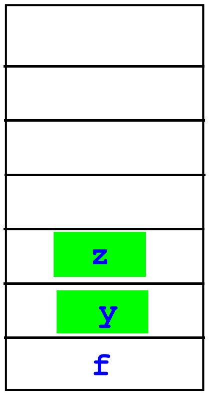
Line Editing

- Initialise a new stack.
- For each character read:
  - if it is a backspace, pop out last char entered
  - if not a backspace, push the char into stack
- To print in reverse, pop out each char for output.

Input : `fgh←r←←yz`

Corrected Input : `fyz`

Reversed Output : `zyf`



Stack

Bracket Matching Problem

Ensures that pairs of brackets are properly matched.

- An Example: `{a,(b+f[4])*3,d+f[5]}`

- Bad Examples:

```
(..)..)       // too many closing brackets

(..(..)       // too many open brackets

[..(..]..)       // mismatched brackets
```

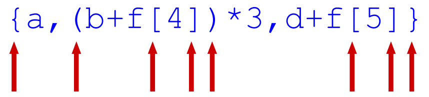
Informal Procedure

Bracket Matching

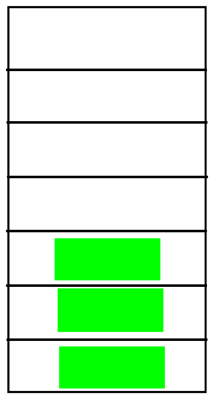
Initialise the stack to empty.

For every char read.
- if open bracket then push onto stack
- if close bracket, then
  - topAndPop from the stack
  - if doesn't match then flag error
- if non-bracket, skip the char read

Example



`{a,(b+f[4])*3,d+f[5]}`

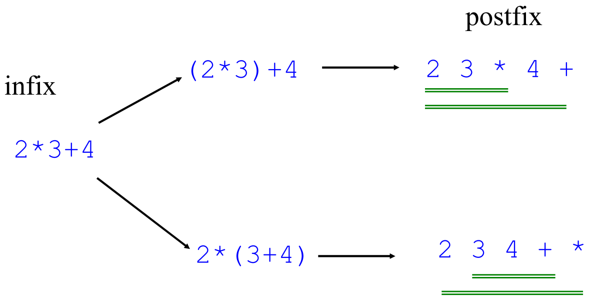


Stack

Postfix Calculator

Computation of arithmetic expressions can be efficiently carried out in Postfix notation with the help of a stack.

Infix	-	`arg1` **`op`** `arg2`
Prefix	-	**`op`** `arg1 arg2`
Postfix	-	`arg1 arg2` **`op`**



postfix

infix

`(2*3)+4`

`2 3 * 4 +`

`2*3+4`

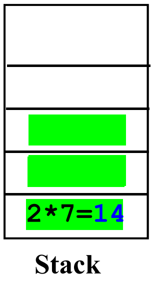
`2*(3+4)`

`2 3 4 + *`

Informal Procedure Postfix Calculator

```
Initialise stack
For each item read.
    If it is an operand,
        push on the stack
    If it is an operator,
        pop arguments from stack;
        perform operation;
        push result onto the stack
```

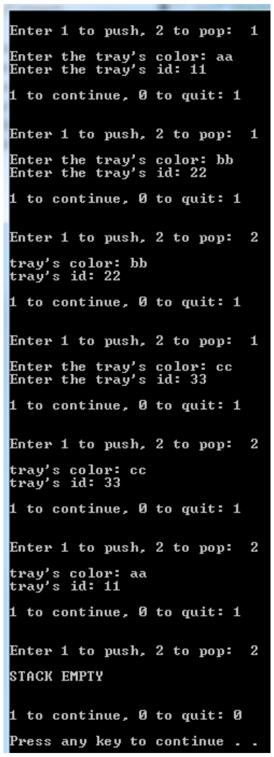
```
Expr
2          s.push(2)
3          s.push(3)
4          s.push(4)
+          arg2=s.topAndPop()
           arg1=s.topAndPop()
           s.push(arg1+arg2)
*          arg2=s.topAndPop()
           arg1=s.topAndPop()
           s.push(arg1*arg2)
```



stack1.c

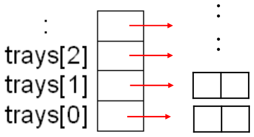
Array of pointers

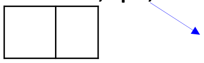
```
typedef struct {
  char color[10]; /* its color */
  int  id;        /* its unique id number */
} node;

// (10 + 2) + 4 = 16 bytes
```



```
Enter 1 to push, 2 to pop:  1
Enter the tray's color: aa
Enter the tray's id: 11
1 to continue, 0 to quit: 1

Enter 1 to push, 2 to pop:  1
Enter the tray's color: bb
Enter the tray's id: 22
1 to continue, 0 to quit: 1

Enter 1 to push, 2 to pop:  2
tray's color: bb
tray's id: 22
1 to continue, 0 to quit: 1

Enter 1 to push, 2 to pop:  1
Enter the tray's color: cc
Enter the tray's id: 33
1 to continue, 0 to quit: 1

Enter 1 to push, 2 to pop:  2
tray's color: cc
tray's id: 33
1 to continue, 0 to quit: 1

Enter 1 to push, 2 to pop:  2
tray's color: aa
tray's id: 11
1 to continue, 0 to quit: 1

Enter 1 to push, 2 to pop:  2
STACK EMPTY
1 to continue, 0 to quit: 0
Press any key to continue . .
```

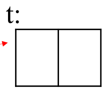
```
// stack1.c - LIFO stack

#include <stdio.h>

#define  SIZE  100

typedef struct {
   char  color[ 10 ];   /* its color */
   int   id;            /* its unique id number */
} node;

node *trays [SIZE];     /* array to hold up to SIZE pointers to node */
int   top = -1;         /* index into the top of stack */
void  get_data(node *), put_data(const node *);
node *pop(), *push(node);
main()
{
    int ans, flag;
    node t, *ptr;
```

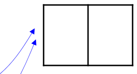




P.T.O.

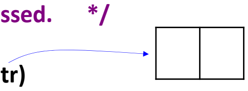
```
/* do stack operations until user signals halt */
  do {
       do {
           printf( "\nEnter 1 to push, 2 to pop:  " );
           scanf( "%d", &ans );
           switch ( ans ) {
              case 1:  /* get a node and add it to stack */
                 get_data( &t );
                 if ( push( t ) == NULL )
                        printf( "\nSTACK FULL\n\n" );
                 break;
              case 2:  /* delete a node from stack and print it */
                 if ( ( ptr = pop() ) != NULL )
                    put_data( ptr );
                 else
                    printf( "\nSTACK EMPTY\n\n" );
                 break;
              default:
                 printf( "\nIllegal response\n" );
                 break;
           }
       } while ( ans != 1 && ans != 2 );
       printf( "\n1 to continue, 0 to quit: " );
       scanf( "%d", &flag );
       printf( "\n" );
  } while ( flag );
  return 0;
}
```



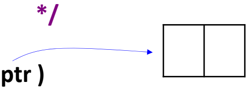


P.T.O.

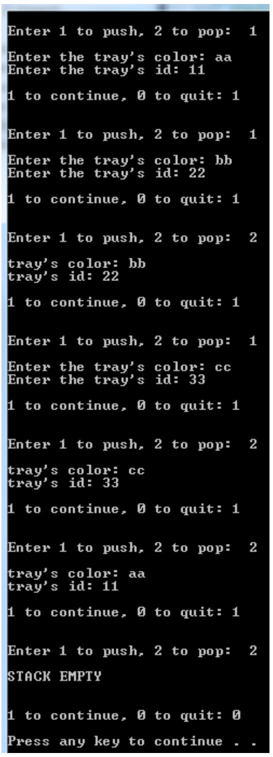
```
/*  get_data prompts the user for a tray's color and id and stores
    it at the address passed.     */

void  get_data(node *ptr)
{
    printf( "\nEnter the tray's color: " );
    scanf( "%s", ptr -> color );
    printf( "Enter the tray's id: " );
    scanf( "%d", & (ptr -> id) );
}
```



```
/*  put_data writes the color and id of the node whose
    address is passed by ptr.     */

void put_data( const node *ptr )
{
    printf( "\ntray's color: %s\n", ptr -> color );
    printf( "tray's id: %d\n", ptr -> id );
}
```

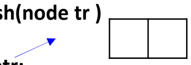


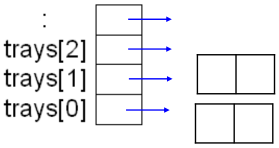
P.T.O.



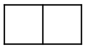
```
Enter 1 to push, 2 to pop:  1

Enter the tray's color: aa
Enter the tray's id: 11

1 to continue, 0 to quit: 1


Enter 1 to push, 2 to pop:  1

Enter the tray's color: bb
Enter the tray's id: 22

1 to continue, 0 to quit: 1


Enter 1 to push, 2 to pop:  2

tray's color: bb
tray's id: 22

1 to continue, 0 to quit: 1


Enter 1 to push, 2 to pop:  1

Enter the tray's color: cc
Enter the tray's id: 33

1 to continue, 0 to quit: 1


Enter 1 to push, 2 to pop:  2

tray's color: cc
tray's id: 33

1 to continue, 0 to quit: 1


Enter 1 to push, 2 to pop:  2

tray's color: aa
tray's id: 11

1 to continue, 0 to quit: 1


Enter 1 to push, 2 to pop:  2

STACK EMPTY


1 to continue, 0 to quit: 0

Press any key to continue . .
```

```
/*  If the stack is full, push returns NULL. Otherwise, push
    allocates storage for a node, copies the data passed into the
    allocated storage, pushes a pointer to the node onto the stack,
    and returns the address of the node added. */
node  *push(node tr )
{
    node *ptr;
    if ( top  >=  SIZE - 1 )   /* stack full? */
        return NULL;
    ptr = (node *) malloc( sizeof (node) );   /* new node */
    *ptr = tr;                  /* store data */
    trays[ ++top ] = ptr;  /* push it and update top */
    return ptr;
}
```

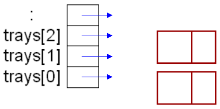




```
/*  If the stack is empty, pop returns NULL. Otherwise, pop copies
    the top node to permanent storage, frees the stack storage,
    updates top, and returns the address of the popped node.   */
node *pop( void )
{
    static node this1;
    if ( top < 0 ) /* empty stack? */
        return NULL;
    this1 = *trays[ top ];   /* copy top node */
    free( trays[ top-- ] );   /* collect garbage */
    return &this1;
}
```



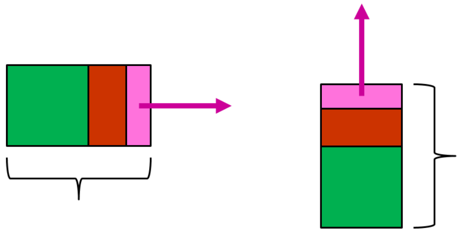
```
// stack1.c - LIFO stack
#include <stdio.h>
#define  SIZE  100
typedef struct {
   char  color[ 10 ];    /* its color */
   int   id;             /* its unique id number */
} node;
node *trays [SIZE];  /* array to hold up to SIZE pointers to node */
int   top = -1;          /* index into the top of stack */
void  get_data(node *), put_data(const node *);
node *pop(), *push(node);
main()
{
     int ans, flag;
      node t,                                        *ptr;
     // clrscr();
     /* do stack operations until user signals halt */
     do {
             do {
                     printf( "\nEnter 1 to push, 2 to pop:  " );
                     scanf( "%d", &ans );
             switch ( ans ) {
                  case 1:  /* get a node and add it to stack */
                      get_data( &t );
                      if ( push( t ) == NULL )
                          printf( "\nSTACK FULL\n\n" );
                                   break;
                   case 2:  /* delete a node from stack and print it */
                       if ( ( ptr = pop() ) != NULL )
                           put_data( ptr );
                       else
                           printf( "\nSTACK EMPTY\n\n" );
                       break;
                    default:
                        printf( "\nIllegal response\n" );
                       break;
                    }
                    } while ( ans != 1 && ans != 2 );
          printf( "\n1 to continue, 0 to quit: " );
          scanf( "%d", &flag );
          printf( "\n" );
     } while ( flag );
     return 0;
}
```

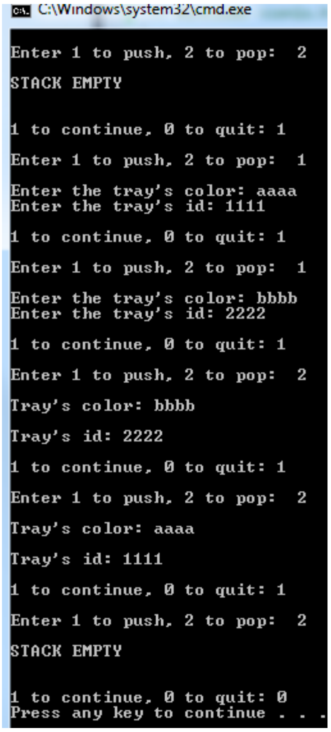


```
/*   get_data prompts the user for a tray's color and id and stores
     it at the address passed.      */
void  get_data(node *ptr)
{
     printf( "\nEnter the tray's color: " );
     scanf( "%s", ptr -> color );
     printf( "Enter the tray's id: " );
     scanf( "%d", & (ptr -> id) );
}

/*   put_data writes the color and id of the node whose address is
     passed by ptr.      */
void put_data( const node *ptr )
{
     printf( "\ntray's color: %s\n", ptr -> color );
     printf( "tray's id: %d\n", ptr -> id );
}

/*   If the stack is full, push returns NULL. Otherwise, push
     allocates storage for a node, copies the data passed into the
     allocated storage, pushes a pointer to the node onto the stack,
     and returns the address of the node added. */
node  *push(node tr )
{
     node *ptr;
     if ( top  >=  SIZE - 1 ) /* stack full? */
             return NULL;
     ptr = (node *) malloc( sizeof (node) ); /* new node */
     *ptr = tr;             /* store data */
     trays[ ++top ] = ptr;  /* push it and update top */
     return ptr;
}

/*   If the stack is empty, pop returns NULL. Otherwise, pop copies
     the top node to permanent storage, frees the stack storage,
     updates top, and returns the address of the popped node.   */
node *pop( void )
{
     static node this1;
     if ( top < 0 ) /* empty stack? */
             return NULL;
     this1 = *trays[ top ];   /* copy top node */
     free( trays[ top-- ] );  /* collect garbage */
     return &this1;
}
```

stack2.c

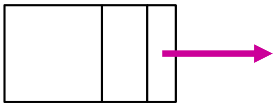
```
typedef struct tray {
  char  color[10];
  int   id;
  struct tray *below;
     /* pointer to successor on stack */
} node;
```



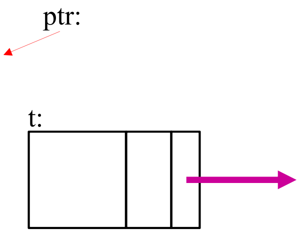


```
C:\Windows\system32\cmd.exe

Enter 1 to push, 2 to pop:  2
STACK EMPTY

1 to continue, 0 to quit: 1
Enter 1 to push, 2 to pop:  1
Enter the tray's color: aaaa
Enter the tray's id: 1111
1 to continue, 0 to quit: 1
Enter 1 to push, 2 to pop:  1
Enter the tray's color: bbbb
Enter the tray's id: 2222
1 to continue, 0 to quit: 1
Enter 1 to push, 2 to pop:  2
Tray's color: bbbb
Tray's id: 2222
1 to continue, 0 to quit: 1
Enter 1 to push, 2 to pop:  2
Tray's color: aaaa
Tray's id: 1111
1 to continue, 0 to quit: 1
Enter 1 to push, 2 to pop:  2
STACK EMPTY

1 to continue, 0 to quit: 0
Press any key to continue . . .
```

```
// stack2.c - LIFO stack
// Implementation of Stack by linked list

#include <stdio.h>

#define SIZE 100

typedef struct tray {
   char  color[ 10 ];       /* its color */
   int   id;                /* its unique id number */
   struct tray *below;   /* pointer to successor on stack */
} node;

node *top = NULL;    /* pointer to top node on stack */
int   currsize = 0;        /* number of items on stack */
void  get_data(node *ptr ), put_data( const node *ptr );
node *pop( void ), *push( node tr );

main()
{
     int ans, flag;
     node t, *ptr;
     // clrscr();
```



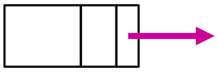
```
/* do stack operations until user signals halt */
do {
        do {
         printf( "\nEnter 1 to push, 2 to pop:  " );
         scanf( "%d", &ans );
         switch ( ans ) {
            case 1: /* get a node and add it to stack */
               get_data( &t );
               if ( push( t ) == NULL )
                    printf( "\nSTACK FULL\n\n" );
               break;
            case 2: /* delete a node from stack and print it */
               if ( ( ptr = pop() ) != NULL )
                  put_data( ptr );
               else
                  printf( "\nSTACK EMPTY\n\n" );
               break;
            default:
               printf( "\nIllegal response\n" );
               break;
         }
        } while ( ans != 1 && ans != 2 );
        printf( "\n1 to continue, 0 to quit: " );
        scanf( "%d", &flag );
} while ( flag );
return 0;
}
```



P.T.O.

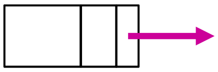
/* get_data prompts the user for a tray's color and id and stores it at the address passed. */

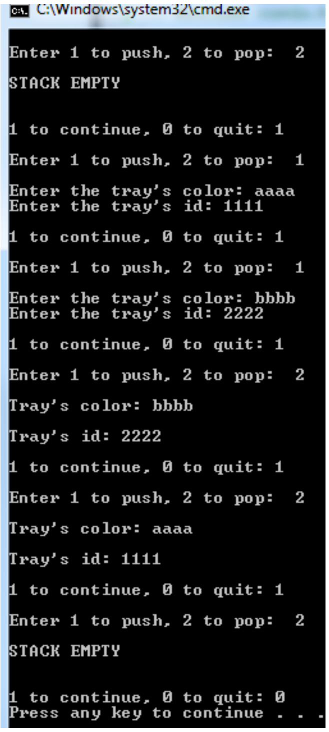
```
void  get_data(node *ptr )
{
    printf( "\nEnter the tray's color: " );
    scanf( "%s", ptr -> color );
    printf( "Enter the tray's id: " );
    scanf( "%d", &( ptr -> id ) );
}
```



/* put_data writes the color and id of the node whose address is passed by ptr. */

```
void put_data( const node *ptr )
{
    printf( "\nTray's color: %s\n", ptr -> color );
    printf( "\nTray's id: %d\n", ptr -> id );
}
```



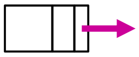


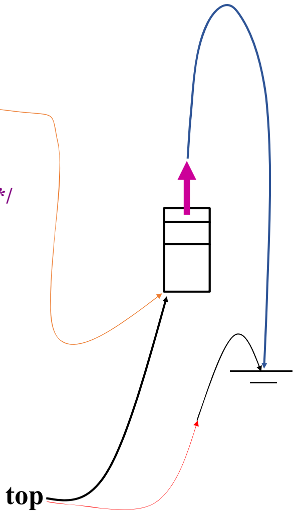
```
Enter 1 to push, 2 to pop:  2

STACK EMPTY

1 to continue, 0 to quit: 1

Enter 1 to push, 2 to pop:  1

Enter the tray’s color: aaaa
Enter the tray’s id: 1111

1 to continue, 0 to quit: 1

Enter 1 to push, 2 to pop:  1

Enter the tray’s color: bbbb
Enter the tray’s id: 2222

1 to continue, 0 to quit: 1

Enter 1 to push, 2 to pop:  2

Tray’s color: bbbb

Tray’s id: 2222

1 to continue, 0 to quit: 1

Enter 1 to push, 2 to pop:  2

Tray’s color: aaaa

Tray’s id: 1111

1 to continue, 0 to quit: 1

Enter 1 to push, 2 to pop:  2

STACK EMPTY

1 to continue, 0 to quit: 0
Press any key to continue . . .
```

P.T.O.

/* If the stack is full, push returns NULL. Otherwise, push allocates storage for a node, copies the data passed into the allocated storage, adds the node to the linked list, updates top and the current size of the stack, and returns the address of the node added. */

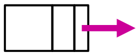
```
node  *push(node tr )
{
    node *ptr;
    if ( currsize  >=  SIZE )     /* stack full? */
         return NULL;
    ptr = (node *) malloc( sizeof ( node ) );   /* new TRAY */
    *ptr = tr;                   /* store data */
    ptr -> below = top;   /* push it on stack */
    top = ptr;                  /* update top */
    ++currsize;                /* update current stack size */
    return ptr;
}
```

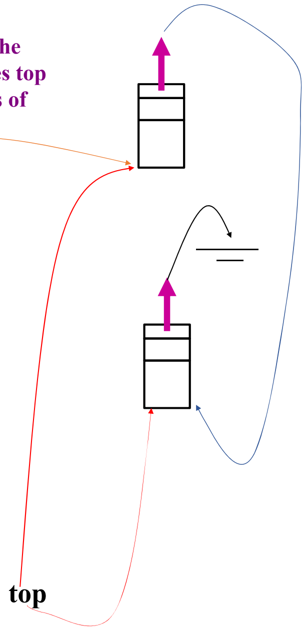




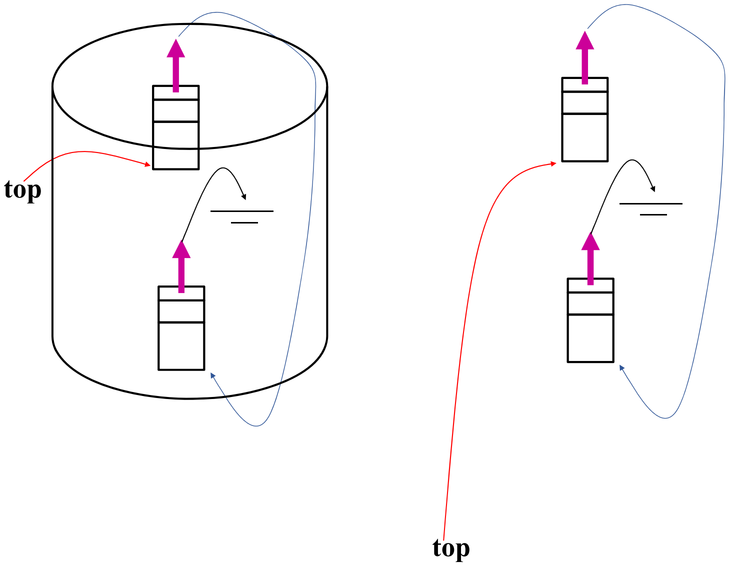
P.T.O.

```
/*   If the stack is full, push returns NULL. Otherwise, push
     allocates storage for a node, copies the data passed into the
     allocated storage, adds the node to the linked list, updates top
     and the current size of the stack, and returns the address of
     the node added. */

node  *push(node tr )
{
     node *ptr;
     if ( currsize  >=  SIZE )      /* stack full? */
          return NULL;
     ptr = (node *) malloc( sizeof ( node ) );   /* new TRAY */
     *ptr = tr;                   /* store data */
     ptr -> below = top;   /* push it on stack */
     top = ptr;                   /* update top */
     ++currsize;                  /* update current stack size */
     return ptr;
}
```

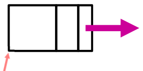


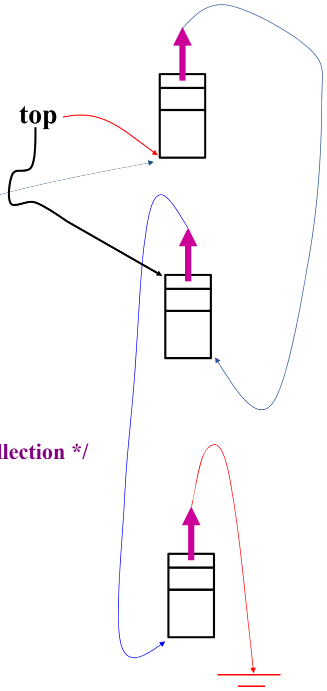


P.T.O.

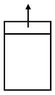


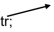
```
/*   If the stack is empty, pop returns NULL. Otherwise, pop copies
     the top node to permanent storage, updates top, frees the stack
     storage, updates the current size of the stack, and returns the
     address of the popped node.   */

node *pop( void )
{
     static node popped_node;
     node *ptr;

     if ( currsize < 1 )          /* empty stack? */
          return NULL;

     popped_node = *top;     /* copy data to be returned */
     ptr = top;                     /* save address of 1st node for garbage collection */
     top = top -> below;      /* update top */
     free( ptr );                   /* collect garbage */
     --currsize;                    /* update current size */
     return &popped_node;
}
```

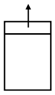


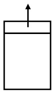


```
// stack2.c - LIFO stack
// Implementation of Stack by linked list
#include <stdio.h>
#define SIZE 100

typedef struct tray {
   char  color[ 10 ];    /* its color */
   int   id;             /* its unique id number */
   struct tray *below;   /* pointer to successor on stack */
} node;
node *top = NULL;    /* pointer to top node on stack */
int   currsize = 0;  /* number of items on stack */
void  get_data(node *ptr ), put_data( const node *ptr );
node *pop( void ), *push( node tr );

main()
{
   int ans, flag;
   node t,                              *ptr;
do {
      do {
         printf( "\nEnter 1 to push, 2 to pop:  " );
         scanf( "%d", &ans );
         switch ( ans ) {
            case 1: /* get a node and add it to stack */
               get_data( &t );
               if ( push( t ) == NULL ) printf( "\nSTACK FULL\n\n" );
               break;
            case 2: /* delete a node from stack and print it */
              if ( ( ptr = pop() ) != NULL )
                 put_data( ptr );
              else
                 printf( "\nSTACK EMPTY\n\n" );
              break;
            default:
              printf( "\nIllegal response\n" );
              break;
           }
       } while ( ans != 1 && ans != 2 );
     printf( "\n1 to continue, 0 to quit: " );
      scanf( "%d", &flag );
   } while ( flag );
  return 0;
}
```

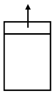


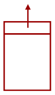






```
/*   get_data prompts the user for a tray's color and id and stores
     it at the address passed.   */
void  get_data(node *ptr )
{
     printf( "\nEnter the tray's color: " );
     scanf( "%s", ptr -> color );
     printf( "Enter the tray's id: " );
     scanf( "%d", &( ptr -> id ) );
}
/*   put_data writes the color and id of the node whose address is
     passed by ptr.   */
void put_data( const node *ptr )
{
     printf( "\nTray's color: %s\n", ptr -> color );
     printf( "\nTray's id: %d\n", ptr -> id );
}
node  *push(node tr )
{
     node *ptr;
     if ( currsize  >=  SIZE ) /* stack full? */
              return NULL;
     ptr = (node *) malloc( sizeof ( node ) ); /* new TRAY */
     *ptr = tr;   /* store data */
     ptr -> below = top; /* push it on stack */
     top = ptr;   /* update top */
     ++currsize;   /* update current stack size */
     return ptr;
}
/*   If the stack is empty, pop returns NULL. Otherwise, pop copies
     the top node to permanent storage, updates top, frees the stack
     storage, updates the current size of the stack, and returns the
     address of the popped node.   */
node *pop( void )
{
     static node popped_node;
     node *ptr;
     if ( currsize < 1 )   /* empty stack? */
              return NULL;
     popped_node = *top;    /* copy data to be returned */
     ptr = top;       /* save address of 1st node for garbage collection */
     top = top -> below;    /* update top */
     free( ptr );           /* collect garbage */
     --currsize;            /* update current size */
     return &popped_node;
}
```





Tutorial 6

Q14.

```
struct mystruct {  int a[5]; };

void f(struct mystruct);
void g(struct mystruct *);

int main() {
    struct mystruct p1, p2;
    p1.a[0] = p2.a[0] = 0;
    f(p1); g(&p2);
    printf("%d %d\n", p1.a[0], p2.a[0]);
}

void f(struct mystruct p) { p.a[0] = 1; }
void g(struct mystruct *p) { p->a[0] = 1; }
```

Answer:

Tutorial 6

Q15.

```
struct mystruct {
    int value;
    struct mystruct *ptr;
};

int dosomething(struct mystruct *);

int main()
{
    int i, j, k;
    struct mystruct p1, p2;

    p1.value = 22; p1.ptr = &p2;
    p2.value = 33; p2.ptr = &p1;
    i = dosomething(&p2);
    j = (p1.ptr == NULL) ? 0 : 1;
    k = (p2.ptr == NULL) ? 0 : 1;
    printf("%d %d %d", i, j, k);
}

int dosomething(struct mystruct *p) {
    int count = 0;
    struct mystruct *q;
    while (p != NULL) {
        count += p->value;
        q = p->ptr;
        p->ptr = NULL;
        p = q;
    }
    return count;
}
```

Answer: