List, Stack and Queue Implementation

Roy Chan

CSC2100B Data Structures Tutorial 2 (Version 2)

January 21, 2009

1. CSC2100B Online Judge Score
2. Structure
3. Linked List
   - Overview
   - Implementation
4. Stack
   - Overview
   - Implementation
5. Queue
   - Overview
   - Implementation

CSC2100B Online Judge Score

```
/* To be reconfirmed                              */
/* Will be adjusted for each assignment           */
/* Note: The score is for reference only,         */
/*       and does not reflect your final grade */

int P, Score;
int award[5]={3,2,1,0,0};

if (in_time) {
    P = num_submission - 1 - award[days_used];
    Score = 30 + (70 - 5*P);
}
else
    Score =0;
```

Structure

- A collection of values (members)
  - Like a class in java or C++, but without methods.

```
struct time {
  int hh;
  int mm;
  int ss;
};
...
struct time t1;
t1.hh=20;
t1.mm=12;
t1.ss=30;
...
```

Structure

- We can also use pointer to structure.

```
struct time {
  int hh;
  int mm;
  int ss;
};
struct time *t1;
(*t1).hh=20;
```

- Pointer to structure is very common, so we gave it a short hand. The above is equivalent to:

```
struct time *t1;
t1->hh=20;  /* Same as (*t1).hh=20; */
```

Structure

- Allow us to define alias for a data type.

```
typedef int My_integer_type;
...
My_integer_type x=3;
```

- Typedef can be used for structures.

```
typedef struct {
  int hh;
  int mm;
  int ss
} Time_type;
...
Time_type t1;
T1.hh=12;
...
```

Dynamic Memory Allocations

- We can allocate memory at run time using malloc.
    - malloc can be used to allocate a piece of memory of the specified size, and returns a pointer to it.
- Example:

```
Time_type *t1;
t1 = (Time_type*)malloc(sizeof(Time_type));
```

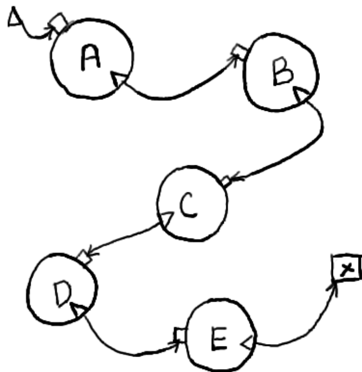
1. Allocate enough memory for storing a Time_type variable (which is a structure in this case).
2. Return a pointer to it.
3. Cast it to a pointer to Time_type, and assign it (the pointer) to t1

Dynamic Memory Allocations

- Use free to de-allocate the memory when it is no longer needed.
- This is important because there is no garbage collection in C. So you will run out of memory if you keep allocating without de-allocating. ("Memory Leak")

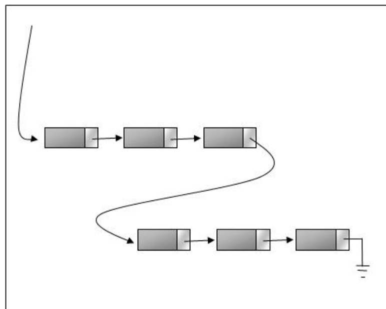
```
Time_type *t1;
t1 = (Time_type*)malloc(sizeof(Time_type));
...
T1->hh=12;
/* some interesting works */
...
free((void *)t1);
/* De-allocate when we no longer need it*/
```

Linked List



Linked List Overview

- A list of structures (nodes)
- Each structure contains
  - Element (to store data)
  - Pointer to next structure
- Insert()
- Delete()
- Print()



Linked List Implementation

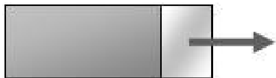
- ```
  struct node_s {
    int data;
    struct node_s *next;
  };
  typedef struct node_s node;
  ```

  Also works with char, double, etc.
- To create a node variable

  ```
  node anode;   //Allocated in compile time
  ```

  or

  ```
  node *anode = (node *) malloc(sizeof(node)); //Dynamic
  ```

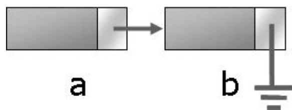

  ```

Linked List Implementation

- Link two nodes together

```
node a, b;
a.next = &b;
b.next = NULL;

/* Pointer to node version /
node *a, *b;
a = (node *)malloc(sizeof(node));
b = (node *)malloc(sizeof(node));
b->next = NULL;
a->next = b;
```



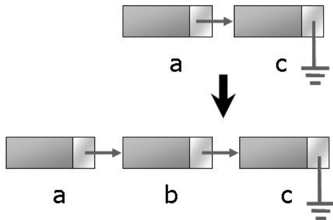
Linked List Implementation

- Insert a node to a list

```
node a, b, c;
c.next = NULL;

/* Original, only a and c */
a.next = &c;

/* New, with b between a and c */
b.next = &c;
a.next = &b;
```



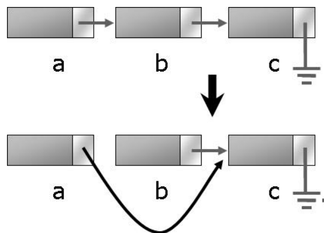
Linked List Implementation

- Delete a node from a list

```
node a, b, c;
c.next = NULL;

/* Original*/
a.next = &b;
b.next = &c;

/* New, b is removed from the list */
a.next = &c;
```

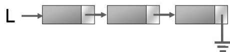


Linked List Implementation

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

// Create a list first
node *L = (node *)malloc(sizeof(node));
node *p;
L->data = 0;
p = L;
for (x=1 ; x<=num ; x++){
  p->next = (node *)malloc(sizeof(node));
  p = p->next;
  p->data = x;
}
p->next = NULL;

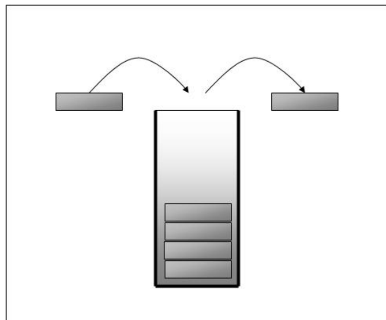
//And then print it
p = L;
while (p != NULL) {
  printf("%d ", p->data);
  p = p->next;
}
putchar('\n');
```



Stack

Stack Overview

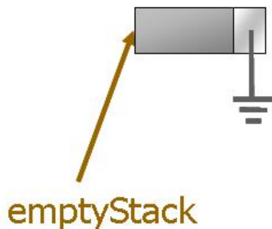
- Last In First Out (LIFO)
- Push()
- Pop()
- Top()
- Isempty()



Stack Implementation

- Can be implemented by linked list or array
- Create an empty stack

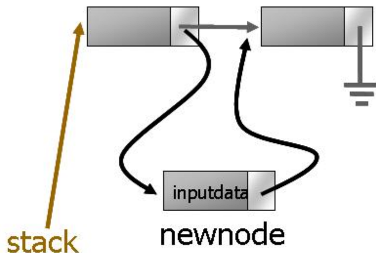
```
node *create(void) {
  node *emptyStack;
  emptyStack = (node *) malloc(sizeof(node));
  emptyStack->next = NULL;
  return emptyStack;
}
```



Stack Implementation

- Push an entry into the stack

```
void Push(int inputdata, node *stack) {
  node *newnode = (node *)malloc(sizeof(node));
  newnode->data = inputdata;
  newnode->next = stack->next;
  stack->next = newnode;
}
```

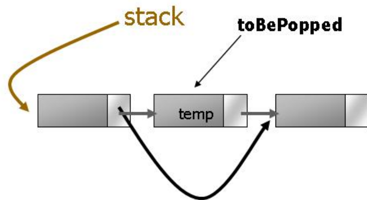


Stack Implementation

- Pop an entry from the stack

```
int Pop(node *stack) {
  int temp;
  node *toBePopped;

  if (stack->next != NULL) {
    temp = stack->next->data;
    toBePopped = stack->next;
    stack->next = stack->next->next;
    free(toBePopped);
    return temp;
  }
  else return NULL;
}
```



Stack Implementation

- Return the top element in the stack

```
int top(node *stack) {
  if (stack->next != NULL)
    return stack->next->data;
  else return NULL;
}
```

- Determine if the stack is empty

```
int is_empty(node *stack) {
  return (stack->next == NULL);
}
```

Stack Implementation

```
//Example Application

void main() {
  node *mystack = create_stack();
  Push(1, mystack);
  Push(2, mystack);
  Push(3, mystack);
  while (!is_empty(mystack)) {
    printf("%d\n", Pop(mystack));
  }
}
```

Stack Implementation

```
#include <stdio.h>
#define NULL 0

struct node_s {

      int data;
      struct node_s *next;

};

typedef struct node_s node;

node *create_stack(void) {
 node *emptyStack;
 emptyStack=(node *)malloc(sizeof  node));
 emptyStack->next = NULL;
 return emptyStack;
}

void Push(int inputdata, node *stack) {
      node *newnode = (node *)malloc(sizeof(node));
      newnode->data = inputdata;
      newnode->next = stack->next;
      stack->next = newnode;
}
```

```
int Pop(node *stack) {
      int temp;
      node *toBePopped;
      if (stack->next != NULL) {
            temp = stack->next->data;
            toBePopped = stack->next;
            stack->next = stack->next->next;
            free(toBePopped);
            return temp;
      }
      else return NULL;
}

int top(node *stack) {
      if (stack->next != NULL)
            return stack->next->data;
      else return NULL;
}

int is_empty(node *stack) {
      return (stack->next == NULL);
}

int Main() {
      node *mystack = create_stack();
      Push(1, mystack);
      Push(2, mystack);
      printf("%d", Pop(mystack));
      ... etc ....
}
```

Stack Implementation

```
/* stack.c */
#include <stdio.h>
#include "stack.h"

void Push(int inputdata, node *stack) {
        node *newnode = (node *)malloc(sizeof(     node));
        newnode->data = inputdata;
        newnode->next = stack->next;
        stack->next = newnode;
}
int Pop(node *stack) {
        int temp;
        node *toBePopped;
        if (stack->next != NULL) {
                temp = stack->next->data;
                toBePopped = stack->next;
                stack->next = stack->next->next;
                free(toBePopped);
                return temp;
        }
        else return NULL;
}

int top(node *stack) {
        if (stack->next != NULL)
                return stack->next->data;
        else return NULL;
}

int is_empty(node *stack) {
        return (stack->next == NULL);
}
```

```
/* Stack.h */

struct node_s {
  int data;
  struct node_s *next;
};

typedef struct node_s node;

node *create_stack(void);
void Push(int inputdata, node *stack);
int Pop(node *stack) ;
int top(node *stack) ;
int is_empty(node *stack) ;
```

Stack Implementation

```
#include <stdio.h>
#include "stack.h"

int main() {
  node *mystack = create_stack();
  Push(1, mystack);
  Push(2, mystack);
  Push(3, mystack);
  while (!is_empty(mystack)) {
     printf("%d\n", Pop(mystack));
  }
  return 0;
}
```

Stack Implementation using Array

- Implement a stack using array

```
typedef struct
{
  int *data; //data is an array of integer
  int top; //position of top element
  int size; //maximum number of data in the stack
}Stack;
```

Stack Implementation using Array

- createStack, makeEmtpy

```
//return 1 for success, 0 for fail
int createStack(Stack *astack, int size)
{
  astack->data = (int*)malloc(sizeof(int)*size);
  if (astack->data==NULL)  // Malloc failed
    return 0;

  astack->size = size;
  astack->top = -1;

  return 1;
}

void makeEmpty(Stack *astack)
{
  astack->top = -1;
}
```

Stack Implementation using Array

- isEmpty, isFull

```
int isEmpty(Stack *astack)
{
  if (astack->top < 0)
    return 1;
  else
    return 0;
}

int isFull(Stack *astack)
{
  if (astack->top >= astack->size-1)
    return 1;
  else
    return 0;
}
```

Stack Implementation using Array

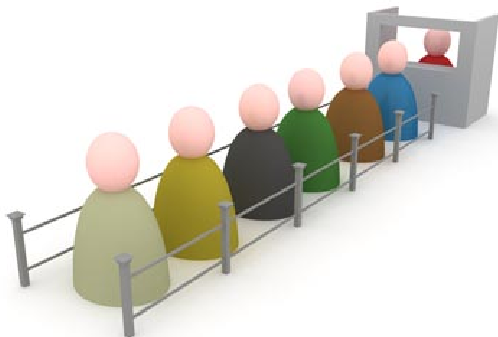
- top, pop, push

```
int top(Stack *astack)
{
  return astack->data[astack->top];
}

int pop(Stack *astack)
{
  int adata = top(astack);
  astack->top--;
  return adata;
}

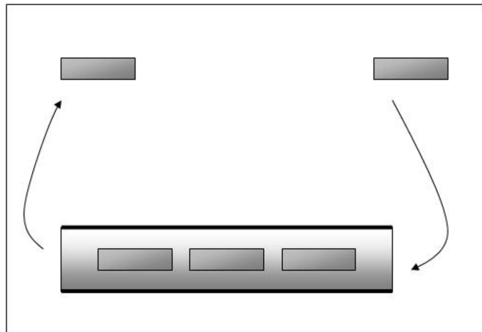
void push(Stack *astack, int adata)
{
  astack->top++;
  astack->data[astack->top] = adata;
}
```

Queue



Queue Overview

- First In First Out (FIFO)
- Enqueue
- Dequeue



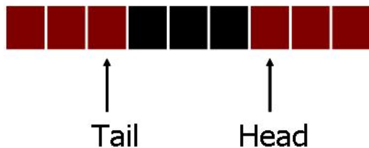
Queue Implementation

- A queue may be implemented using linked-list or array
- Implement a queue using array

Queue Implementation

- Implementing a queue using circular array

```
typedef struct {
  int *data; //data is an array of int
  int head;
  int tail;
  int num; // number of elements in queue
  int size;// size of queue
}Queue;
```



Queue Implementation

- createQueue

```
//return 1 for success, 0 for fail
int createQueue(Queue aqueue, int size)
{
  aqueue->data = (int*)malloc(sizeof(int)*size);
  if (aqueue->data == NULL)
    return 0;

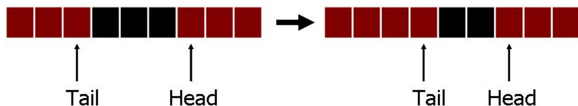
  aqueue->head = 0;
  aqueue->tail = -1;
  aqueue->num = 0;
  aqueue->size = size;

  return 1;
}
```

Queue Implementation

- enqueue

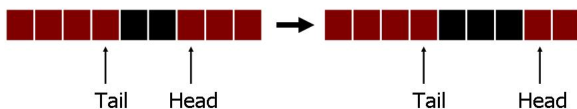
```
void enqueue(Queue *aqueue, int adata)
{
  aqueue->tail = (aqueue->tail + 1) % aqueue->size;
  aqueue->data[queue->tail] =adata;
  aqueue->num++;
}
```



Queue Implementation

- dequeue

```
queue_data dequeue(Queue *aqueue)
{
  int adata = aqueue->data[aqueue->head];
  aqueue->head = (aqueue->head + 1) % aqueue->size;
  aqueue->num--;
  return adata;
}
```



Queue Implementation

- isEmpty, isFull

```
int isEmpty(Queue *aqueue) {
  return (aqueue->num == 0);
}

int isFull(Queue *aqueue) {
  return (aqueue->num == aqueue->size);
}
```

Queue Implementation

- front, makeEmpty

```
int front(Queue *aqueue)
{
  return aqueue->data[aqueue->head];
}

void makeEmpty(Queue *aqueue)
{
  aqueue ->head = 0;
  aqueue ->tail = -1;
  aqueue ->num = 0;
}
```

Question