```
/* istack.h */

#ifndef _istack_h
#define _istack_h

/* pointer to an undefined struct provides
*  small amount of data hiding here.
*/

typedef struct istack_s *istack;

istack istack_init(unsigned int capacity);
int istack_isfull();
int istack_isempty();
void istack_push(istack s, int item);
int istack_pop(istack s);
void istack_destroy(istack s);

#endif
```

```
/************************************************************/

/* istacktest.c */

#include <stdlib.h>
#include <stdio.h>
#include "istack.h"


int main(){

  istack s = istack_init(100);
  for(int i=0; i<10; i++){
    istack_push(s, i);
  }

  while (!istack_isempty(s)){
    printf("%d\n", istack_pop(s));
  }

  istack_destroy(s);
  return 0;
}
```

```
/************************************************************/
/* istack.c */

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

#include "istack.h"

struct istack_s {
  unsigned int capacity;
  unsigned int count;
  int *items;
};

istack istack_init(unsigned int capacity)
{

  istack s = malloc(sizeof(struct istack_s));
  if (s == NULL){
    printf("Failed to allocate stack memory\n");
    exit(1);
  }

  s->items = malloc(capacity*sizeof(int));
  if (s->items == NULL){
    printf("Failed to allocate stack memory\n");
    exit(1);
  }
  s->capacity = capacity;
  s->count = 0;
}

void istack_push(istack s, int item){
  assert(s->count < s->capacity);
  s->items[s->count] = item;
  s->count++;
}

int istack_pop(istack s){
  assert(s->count > 0);
  s->count--;
  return s->items[s->count];
}

int istack_isfull(istack s){
  return s->count == s->capacity;
}

int istack_isempty(istack s){
  return s->count == 0;
}

void istack_destroy(istack s){
  free(s->items);
  free(s);
}
```

```
/************************************************************/
/* teststack.c
 *    Using a generic stack of pointers */

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

int main(){
  int *num;

  stack s = stack_init(100);

  for(int i=0; i<10; i++){
    num = malloc(sizeof(int));
    *num = i;
    stack_push(s, num);
  }

  while (!stack_isempty(s)){
      num = stack_pop(s);
      printf("%d\n", *num);
      free(num);
  }

  stack_destroy(s);

  return 0;
}
```