Computer Programming II

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Advanced Linked List

Advanced Linked List

- Example: advList/advList.h

```
10 /*************************************************************
11  *  Define a structure for linked list elements.
12  *************************************************************/
13 typedef struct ListElmt_ {
14     void               *data;
15     struct ListElmt_   *next;
16 } ListElmt;
```

```
18 /*************************************************************
19  *  Define a structure for linked lists.
20  *************************************************************/
21 typedef struct List_ {
22     int                size;
23     void               (*destroy)(void *data);
24     ListElmt           *head;
25     ListElmt           *tail;
26 } List;
```

Advanced Linked List

- Example: advList/advList.c

```
13 void list_init(List *list, void (*destroy)(void *data)) {
14         /************************************************
15          *  Initialize the list.
16          ************************************************
17         list->size = 0;
18         list->destroy = destroy;
19         list->head = NULL;
20         list->tail = NULL;
21         return;
22 }
```

Advanced Linked List

- Example: advList/advList.c

```
13 void list_init(List *list, void (*destroy)(void *data)) {
14         /************************************************************
15          *  Initialize the list.
16          ************************************************************/
17         list->size = 0;
18         list->destroy = destroy;
19         list->head = NULL;
20         list->tail = NULL;
21         return;
22 }
```

Advanced Linked List

- Example: advList/advList.c

```
13 void list_init(List *list, void (*destroy)(void *data)) {
14         /************************************************************
15          *  Initialize the list.
16          ************************************************************/
17         list->size = 0;
18         list->destroy = destroy;
19         list->head = NULL;
20         list->tail = NULL;
21         return;
22 }
```

use function pointer to pass user-defined destroy function

Advanced Linked List

- Example: advList/advList.c

```
26 void list_destroy(List *list) {
27     void                *data;
28     /************************************************************
29      *  Remove each element.
30      ************************************************************
31     while (list_size(list) > 0) {
32         if (list_rem_next(list, NULL, (void **)&data) == 0 &&
33                 list->destroy != NULL) {
34             /************************************************************
35              *  Call a user-defined function to free dynamically
36              ************************************************************
37             list->destroy(data);
38         }
39     }
40     /************************************************************
41      *  No operations are allowed now, but clear the structure as
42      ************************************************************
43     memset(list, 0, sizeof(List));
44     return;
45 }
```

Advanced Linked List

- Example: advList/advList.c

```
26 void list_destroy(List *list) {
27     void                *data;
28     /************************************************************
29      *  Remove each element.
30      ************************************************************
31     while (list_size(list) > 0) {
32         if (list_rem_next(list, NULL, (void **)&data) == 0 &&
33                 list->destroy != NULL) {
34             /************************************************************
35              *  Call a user-defined function to free dynamically
36              ************************************************************
37             list->destroy(data);
38         }
39     }
40     /************************************************************
41      *  No operations are allowed now, but clear the structure as
42      ************************************************************
43     memset(list, 0, sizeof(List));
44     return;
45 }
```

Advanced Linked List

- Example: advList/advList.c

```
26 void list_destroy(List *list) {
27     void                *data;
28     /************************************************************
29      *  Remove each element.
30      ************************************************************/
31     while (list_size(list) > 0) {
32         if (list_rem_next(list, NULL, (void **)&data) == 0 &&
33                 list->destroy != NULL) {
34             /************************************************************
35              *  Call a user-defined function to free dynamically
36              ************************************************************/
37             list->destroy(data);
38         }
39     }
40     /************************************************************
41      *  No operations are allowed now, but clear the structure as
42      ************************************************************/
43     memset(list, 0, sizeof(List));
44     return;
45 }
```

use data to store the removed info

Advanced Linked List

- Example: advList/advList.c

```
26 void list_destroy(List *list) {
27     void                *data;
28     /************************************************************
29      *  Remove each element.
30      ************************************************************
31     while (list_size(list) > 0) {
32         if (list_rem_next(list, NULL, (void **)&data) == 0 &&
33                 list->destroy != NULL) {
34             /************************************************************
35              *  Call a user-defined function to free dynamically
36              ************************************************************
37             list->destroy(data);
38         }
39     }
40     /************************************************************
41      *  No operations are allowed now, but clear the structure as
42      ************************************************************
43     memset(list, 0, sizeof(List));
44     return;
45 }
```

use data to store the removed info

Advanced Linked List

- Example: advList/advList.c

```
26 void list_destroy(List *list) {
27     void                *data;
28     /************************************************************
29      *  Remove each element.
30      ************************************************************/
31     while (list_size(list) > 0) {
32         if (list_rem_next(list, NULL, (void **)&data) == 0 &&
33                 list->destroy != NULL) {
34             /************************************************************
35              *  Call a user-defined function to free dynamically
36              ************************************************************/
37             list->destroy(data);
38         }
39     }
40     /************************************************************
41      *  No operations are allowed now, but clear the structure as
42      ************************************************************/
43     memset(list, 0, sizeof(List));
44     return;
45 }
```

use data to store the removed info

use user-defined function to free data

Advanced Linked List

- Example: advList/advList.c

```
47 /**************************************************************************
48  *  ---------------------------- list_ins_next ----------------------------
49  **************************************************************************/
50 int list_ins_next(List *list, ListElmt *element, const void *data) {
51     ListElmt           *new_element;
52     /**********************************************************************
53      *  Allocate storage for the element.
54      **********************************************************************/
55     if ((new_element = (ListElmt *)malloc(sizeof(ListElmt))) == NULL)
56         return -1;
57     /**********************************************************************
58      *  Insert the element into the list.
59      **********************************************************************/
60     new_element->data = (void *)data;
61     if (element == NULL) {
62         /******************************************************************
63          *  Handle insertion at the head of the list.
64          ******************************************************************/
65         if (list_size(list) == 0)
66             list->tail = new_element;
67         new_element->next = list->head;
68         list->head = new_element;
```

Advanced Linked List

- Example: advList/advList.c

```
47 /**************************************************************************
48  *  ---------------------------- list_ins_next ----------------------------
49  **************************************************************************/
50 int list_ins_next(List *list, ListElmt *element, const void *data) {
51     ListElmt           *new_element;
52     /*********************************************************************
53      *  Allocate storage for the element.
54      *********************************************************************/
55     if ((new_element = (ListElmt *)malloc(sizeof(ListElmt))) == NULL)
56         return -1;
57     /*********************************************************************
58      *  Insert the element into the list.
59      *********************************************************************/
60     new_element->data = (void *)data;
61     if (element == NULL) {
62         /*****************************************
63          *  Handle insertion at the head of the l
64          *****************************************/
65         if (list_size(list) == 0)
66             list->tail = new_element;
67         new_element->next = list->head;
68         list->head = new_element;
69     } else {
70         /*****************************************************************
71          *  Handle insertion somewhere other than at the head.
72          *****************************************************************/
73         if (element->next == NULL)
74             list->tail = new_element;
75         new_element->next = element->next;
76         element->next = new_element;
77     }
78     /*********************************************************************
79      *  Adjust the size of the list to account for the inserted element.
80      *********************************************************************/
81     list->size++;
```

Advanced Linked List

- Example: advList/advList.c

```
88  int list_rem_next(List *list, ListElmt *element, void **data) {
89      ListElmt           *old_element;
90      /*************************************************************
91       *  Do not allow removal from an empty list.
92       *************************************************************
93      if (list_size(list) == 0)
94          return -1;
95      /*************************************************************
96       *  Remove the element from the list.
97       *************************************************************
98      if (element == NULL) {
99          /*********************************************************
100          *  Handle removal from the head of the list.
101          *********************************************************
102         *data = list->head->data;
103         old_element = list->head;
104         list->head = list->head->next;
105         if (list_size(list) == 1)
106             list->tail = NULL;
107     } else {
```

Advanced Linked List

- Example: advList/advList.c

```
88  int list_rem_next(List *list, ListElmt *element, void **data) {
89      ListElmt            *old_element;
90      /************************************************************
91       *  Do not allow removal from an empty list.                *
92       ************************************************************/
93      if (list_size(list) == 0)
94          return -1;
95      /************************************************************
96       *  Remove the element from the list.                       *
97       ************************************************************/
98      if (element == NULL) {
99          /********************************************************
100          *  Handle removal from the head of the list            *
101          ********************************************************/
102         *data = list->head->data;
103         old_element = list->head;
104         list->head = list->head->next;
105         if (list_size(list) == 1)
106             list->tail = NULL;
107     } else {
```

```
107     } else {
108         /********************************************************
109          *  Handle removal from somewhere other than the head.  *
110          ********************************************************/
111         if (element->next == NULL)
112             return -1;
113         *data = element->next->data;
114         old_element = element->next;
115         element->next = element->next->next;
116         if (element->next == NULL)
117             list->tail = element;
118     }
119     /************************************************************
120      *  Free the storage allocated by the abstract data type.   *
121      ************************************************************/
122     free(old_element);
123     /************************************************************
124      *  Adjust the size of the list to account for the removed  *
125      ************************************************************/
126     list->size--;
```

Advanced Linked List

- Example: advList/advTest.c

```
56      for (i = 10; i > 0; i--) {
57          if ((data = (int *)malloc(sizeof(int))) == NULL)
58              return 1;
59          *data = i;
60          if (list_ins_next(&list, NULL, data) != 0)
61              return 1;
62      }
63
64      print_list(&list);
65      printf("=====================================\n");
```

```
67      element = list_head(&list);
68      for (i = 0; i < 7; i++)
69          element = list_next(element);
70      data = list_data(element);
71
72      fprintf(stdout, "Removing an element after the one containing %03d\n", *data);
73      if (list_rem_next(&list, element, (void **)&data) != 0) /* the removed data wi
74          return 1;
75      print_list(&list);
```

Advanced Stack

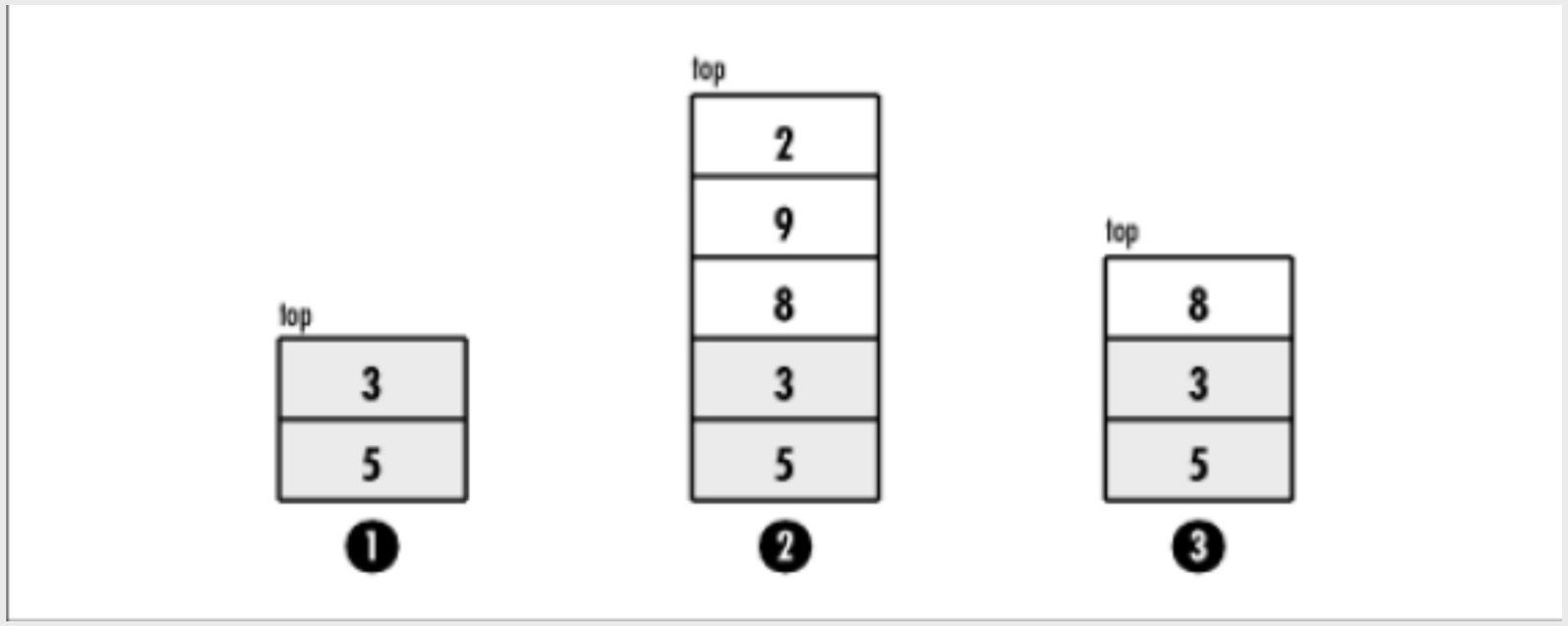
Stacks

- Stack
  - Efficient data structures for storing and retrieving data in a last-in, first-out order
  - Allow us to retrieve data in the opposite order as it was stored
    - New nodes can be added and removed only at the top
    - Similar to a pile of dishes
    - Last-In, First-Out (LIFO)
    - Bottom of stack indicated by a link member to **NULL**
    - Constrained version of a linked list

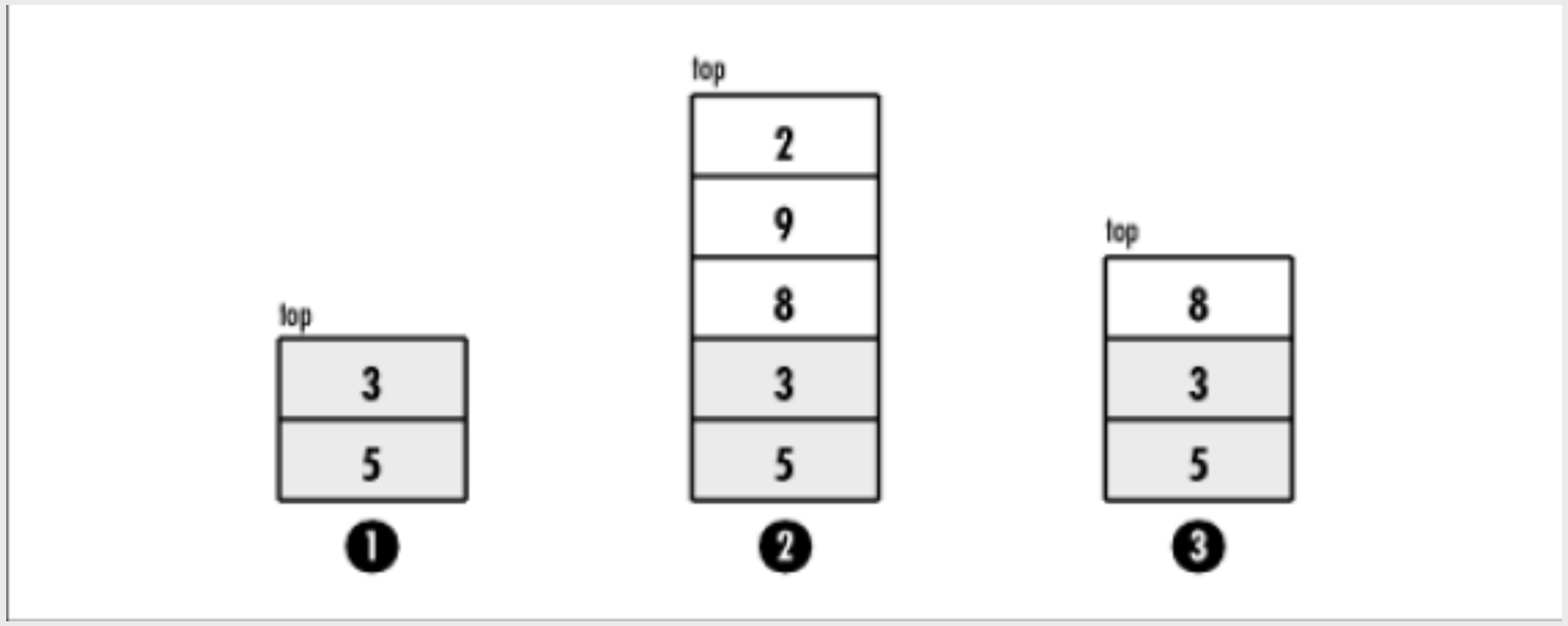
Stacks

- Two main functions in a stack
  - Push
    - Adds a new node to the top of the stack
  - Pop
    - Removes a node from the top
    - Stores the popped value
    - Returns **true** if **pop** was successful

Stacks

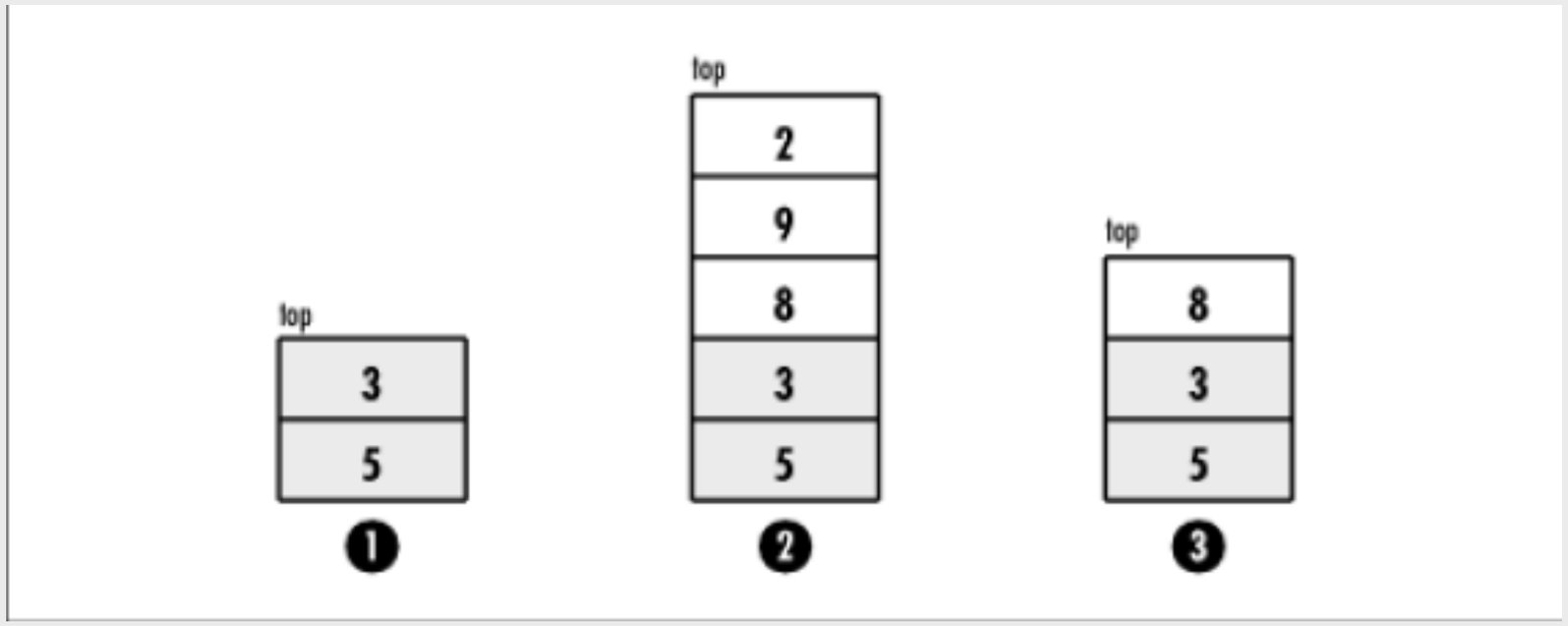


Stacks



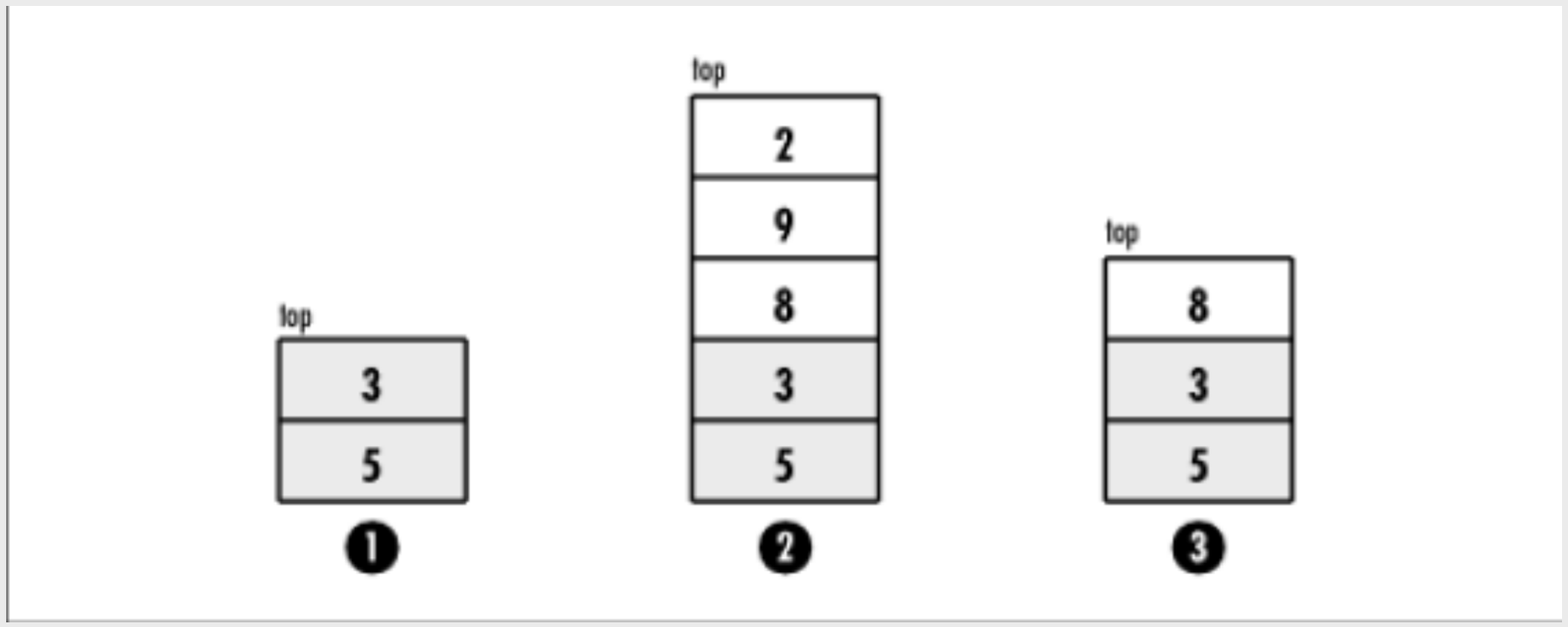
- a stack with some elements (3, 5)

Stacks



- a stack with some elements (3, 5)
- after pushing 8, 9, and 2

Stacks



- a stack with some elements (3, 5)
- after pushing 8, 9, and 2
- after popping 2 and 9

Implementation of Stack

- Example: stack/stack.h

```
 5 #ifndef STACK_H
 6 #define STACK_H
 7
 8 #include <stdlib.h>
 9 #include "list.h"
10
11 /************************************************************************
12 *  Implement stacks as linked lists.                                    *
13 ************************************************************************/
14
15 typedef List Stack;
16
17 /************************************************************************
18 *  ------------------------- Public Interface -------------------------  *
19 ************************************************************************/
20
21 #define stack_init list_init
22 #define stack_destroy list_destroy
23 int stack_push(Stack *stack, const void *data);
24 int stack_pop(Stack *stack, void **data);
25 #define stack_peek(stack) ((stack)->head == NULL ? NULL : (stack)->head->data)
26 #define stack_size list_size
27 #endif
```

Implementation of Stack

- Example: stack/stack.h

```
 5 #ifndef STACK_H
 6 #define STACK_H
 7
 8 #include <stdlib.h>
 9 #include "list.h"
10
11 /************************************************************************
12 *  Implement stacks as linked lists.                                    *
13 ************************************************************************/
14
15 typedef List Stack;
16
17 /************************************************************************
18 *  ------------------------- Public Interface -------------------------  *
19 ************************************************************************/
20
21 #define stack_init list_init
22 #define stack_destroy list_destroy
23 int stack_push(Stack *stack, const void *data);
24 int stack_pop(Stack *stack, void **data);
25 #define stack_peek(stack) ((stack)->head == NULL ? NULL : (stack)->head->data)
26 #define stack_size list_size
27 #endif
```

Implementation of Stack

- Example: stack/stack.h

```
 5 #ifndef STACK_H
 6 #define STACK_H
 7
 8 #include <stdlib.h>
 9 #include "list.h"
10
11 /************************************************************************
12 *  Implement stacks as linked lists.                                    *
13 ************************************************************************/
14
15 typedef List Stack;
16
17 /************************************************************************
18 *  ------------------------- Public Interface -------------------------  *
19 ************************************************************************/
20
21 #define stack_init list_init
22 #define stack_destroy list_destroy
23 int stack_push(Stack *stack, const void *data);
24 int stack_pop(Stack *stack, void **data);
25 #define stack_peek(stack) ((stack)->head == NULL ? NULL : (stack)->head->data)
26 #define stack_size list_size
27 #endif
```

Implementation of Stack

- Example: stack/stack.h

```
 5 #ifndef STACK_H
 6 #define STACK_H
 7
 8 #include <stdlib.h>
 9 #include "list.h"
10
11 /************************************************************************
12 *  Implement stacks as linked lists.                                    *
13 ************************************************************************/
14
15 typedef List Stack;
16
17 /************************************************************************
18 *  ------------------------- Public Interface -------------------------  *
19 ************************************************************************/
20
21 #define stack_init list_init
22 #define stack_destroy list_destroy
23 int stack_push(Stack *stack, const void *data);
24 int stack_pop(Stack *stack, void **data);
25 #define stack_peek(stack) ((stack)->head == NULL ? NULL : (stack)->head->data)
26 #define stack_size list_size
27 #endif
```

Implementation of Stack

- Example: stack/stack.h

```
 5 #ifndef STACK_H
 6 #define STACK_H
 7
 8 #include <stdlib.h>
 9 #include "list.h"
10
11 /************************************************************************
12 *  Implement stacks as linked lists.                                    *
13 ************************************************************************/
14
15 typedef List Stack;
16
17 /************************************************************************
18 *  ------------------------- Public Interface -------------------------  *
19 ************************************************************************/
20
21 #define stack_init list_init
22 #define stack_destroy list_destroy
23 int stack_push(Stack *stack, const void *data);
24 int stack_pop(Stack *stack, void **data);
25 #define stack_peek(stack) ((stack)->head == NULL ? NULL : (stack)->head->data)
26 #define stack_size list_size
27 #endif
```

Implementation of Stack

- Example: stack/stack.h

```
 5 #ifndef STACK_H
 6 #define STACK_H
 7
 8 #include <stdlib.h>
 9 #include "list.h"
10
11 /************************************************************************
12 *  Implement stacks as linked lists.                                    *
13 ************************************************************************/
14
15 typedef List Stack;
16
17 /************************************************************************
18 *  ------------------------- Public Interface -------------------------  *
19 ************************************************************************/
20
21 #define stack_init list_init
22 #define stack_destroy list_destroy
23 int stack_push(Stack *stack, const void *data);
24 int stack_pop(Stack *stack, void **data);
25 #define stack_peek(stack) ((stack)->head == NULL ? NULL : (stack)->head->data)
26 #define stack_size list_size
27 #endif
```

Implementation of Stack

- Example: stack/stack.h

```
 5 #ifndef STACK_H
 6 #define STACK_H
 7
 8 #include <stdlib.h>
 9 #include "list.h"
10
11 /************************************************************************
12 *  Implement stacks as linked lists.                                    *
13 ************************************************************************/
14
15 typedef List Stack;
16
17 /************************************************************************
18 *  ------------------------- Public Interface -------------------------  *
19 ************************************************************************/
20
21 #define stack_init list_init
22 #define stack_destroy list_destroy
23 int stack_push(Stack *stack, const void *data);
24 int stack_pop(Stack *stack, void **data);
25 #define stack_peek(stack) ((stack)->head == NULL ? NULL : (stack)->head->data)
26 #define stack_size list_size
27 #endif
```

Implementation of Stack

- Example: stack/stack.h

```
 5 #ifndef STACK_H
 6 #define STACK_H
 7
 8 #include <stdlib.h>
 9 #include "list.h"
10
11 /************************************************************************
12 *  Implement stacks as linked lists.                                    *
13 ************************************************************************/
14
15 typedef List Stack;
16
17 /************************************************************************
18 *  ------------------------- Public Interface -------------------------  *
19 ************************************************************************/
20
21 #define stack_init list_init
22 #define stack_destroy list_destroy
23 int stack_push(Stack *stack, const void *data);
24 int stack_pop(Stack *stack, void **data);
25 #define stack_peek(stack) ((stack)->head == NULL ? NULL : (stack)->head->data)
26 #define stack_size list_size
27 #endif
```

Implementation of Stack

- Example: stack/stack.c

```c
#include <stdlib.h>
#include "list.h"
#include "stack.h"

/****************************************************************************
 *  ---------------------------- stack_push ----------------------------  *
 ****************************************************************************/
int stack_push(Stack *stack, const void *data) {
    return list_ins_next(stack, NULL, data);
}

/****************************************************************************
 *  ---------------------------- stack_pop ----------------------------  *
 ****************************************************************************/
int stack_pop(Stack *stack, void **data) {
    return list_rem_next(stack, NULL, data);
}
```

Implementation of Stack

- Example: stack/stack.c

```
5  #include <stdlib.h>
6  #include "list.h"
7  #include "stack.h"
8
9  /*****************************************************************************
10  *  ---------------------------- stack_push ----------------------------  *
11  *****************************************************************************/
12 int stack_push(Stack *stack, const void *data) {
13     return list_ins_next(stack, NULL, data);
14 }
15
16 /*****************************************************************************
17  *  ---------------------------- stack_pop ----------------------------  *
18  *****************************************************************************/
19 int stack_pop(Stack *stack, void **data) {
20     return list_rem_next(stack, NULL, data);
21 }
```

Implementation of Stack

- Example: stack/stack.c

```
5  #include <stdlib.h>
6  #include "list.h"
7  #include "stack.h"
8
9  /*****************************************************************************
10  *  ---------------------------- stack_push ----------------------------  *
11  *****************************************************************************/
12 int stack_push(Stack *stack, const void *data) {
13     return list_ins_next(stack, NULL, data);
14 }
15
16 /*****************************************************************************
17  *  ---------------------------- stack_pop ----------------------------  *
18  *****************************************************************************/
19 int stack_pop(Stack *stack, void **data) {
20     return list_rem_next(stack, NULL, data);
21 }
```

Implementation of Stack

- Example: stack/stack.c

```
5  #include <stdlib.h>
6  #include "list.h"
7  #include "stack.h"
8
9  /*****************************************************************************
10  *  ---------------------------- stack_push ----------------------------  *
11  *****************************************************************************/
12 int stack_push(Stack *stack, const void *data) {
13     return list_ins_next(stack, NULL, data);
14 }
15
16 /*****************************************************************************
17  *  ---------------------------- stack_pop ----------------------------  *
18  *****************************************************************************/
19 int stack_pop(Stack *stack, void **data) {
20     return list_rem_next(stack, NULL, data);
21 }
```

NULL means to insert into the head of the list

Implementation of Stack

- Example: stack/stack.c

```c
#include <stdlib.h>
#include "list.h"
#include "stack.h"

/*****************************************************************************
 *  ---------------------------- stack_push ----------------------------  *
 *****************************************************************************/
int stack_push(Stack *stack, const void *data) {
    return list_ins_next(stack, NULL, data);
}

/*****************************************************************************
 *  ---------------------------- stack_pop ----------------------------  *
 *****************************************************************************/
int stack_pop(Stack *stack, void **data) {
    return list_rem_next(stack, NULL, data);
}
```

NULL means to insert into the head of the list

Implementation of Stack

- Example: stack/stack.c

```c
#include <stdlib.h>
#include "list.h"
#include "stack.h"

/**************************************************************************
 *  ---------------------------- stack_push ----------------------------  *
 **************************************************************************/
int stack_push(Stack *stack, const void *data) {
    return list_ins_next(stack, NULL, data);
}

/**************************************************************************
 *  ---------------------------- stack_pop ----------------------------  *
 **************************************************************************/
int stack_pop(Stack *stack, void **data) {
    return list_rem_next(stack, NULL, data);
}
```

NULL means to insert into the head of the list

Implementation of Stack

- Example: stack/stack.c

```
5  #include <stdlib.h>
6  #include "list.h"
7  #include "stack.h"
8
9  /****************************************************************************
10  *  ------------------------------ stack_push ------------------------------  *
11  ****************************************************************************/
12 int stack_push(Stack *stack, const void *data) {
13     return list_ins_next(stack, NULL, data);
14 }
15
16 /****************************************************************************
17  *  ------------------------------ stack_pop -------------------------------  *
18  ****************************************************************************/
19 int stack_pop(Stack *stack, void **data) {
20     return list_rem_next(stack, NULL, data);
21 }
```

NULL means to insert into the head of the list

NULL means to remove from the head of the list

Implementation of Stack

- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
39      /*********************************************************************
40       *  Initialize the stack.                                            *
41       *********************************************************************/
42      stack_init(&stack, free);
43
44      /*********************************************************************
45       *  Perform some stack operations.                                   *
46       *********************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

Implementation of Stack

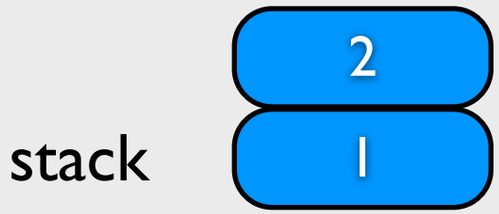
- Example: stack/test.c

```
39  /*****************************************************************************
40   *  Initialize the stack.                                                    *
41   *****************************************************************************/
42  stack_init(&stack, free);
43
44  /*****************************************************************************
45   *  Perform some stack operations.                                           *
46   *****************************************************************************/
47  fprintf(stdout, "Pushing 10 elements\n");
48  for (i = 0; i < 10; i++) {
49      if ((data = (int *)malloc(sizeof(int))) == NULL)
50          return 1;
51      *data = i + 1;
52      if (stack_push(&stack, data) != 0)
53          return 1;
54  }
55  print_stack(&stack);
56
57  fprintf(stdout, "Popping 5 elements\n");
58  for (i = 0; i < 5; i++) {
59      if (stack_pop(&stack, (void **)&data) == 0)
60          free(data);
61      else
62          return 1;
63  }
64  print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

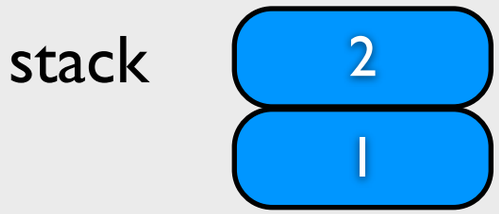
```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

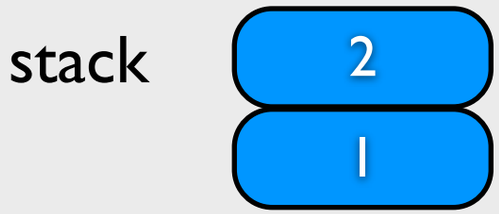
```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

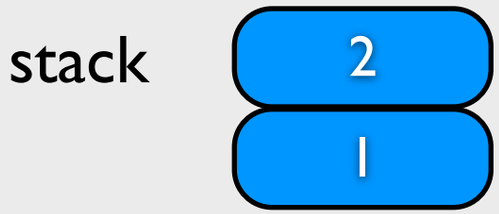
```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

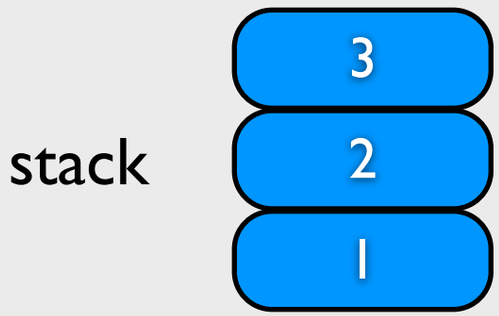
```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

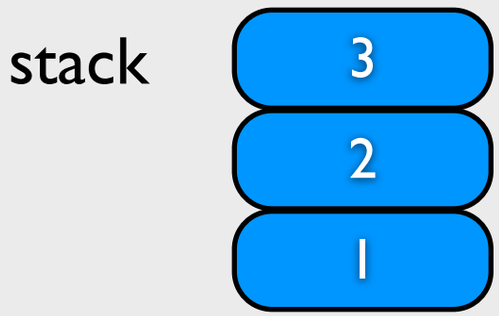
```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

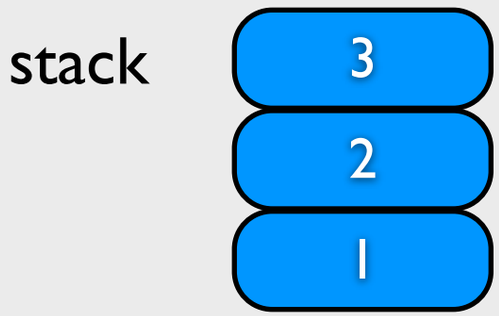
```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

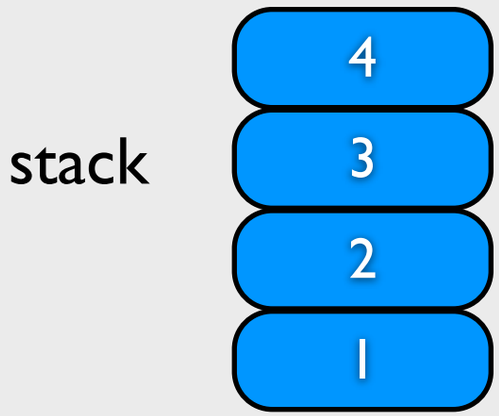
```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

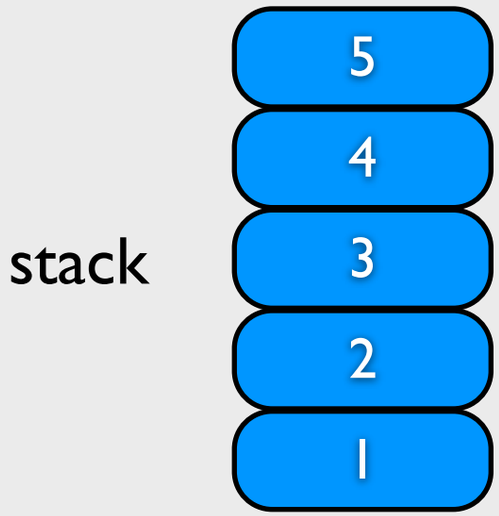
```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

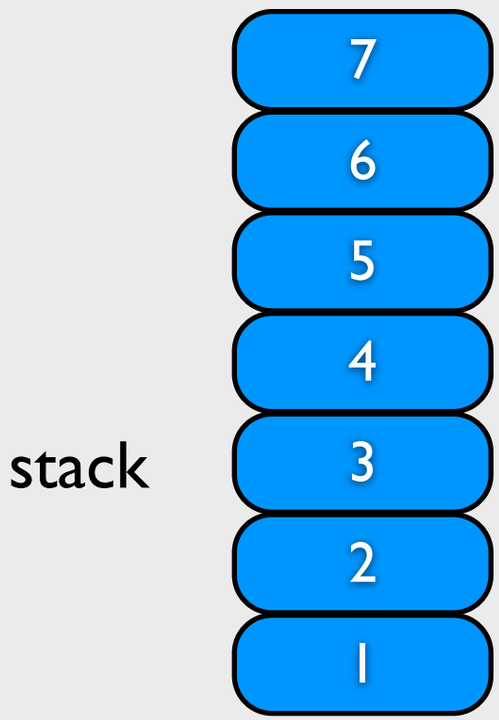
- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

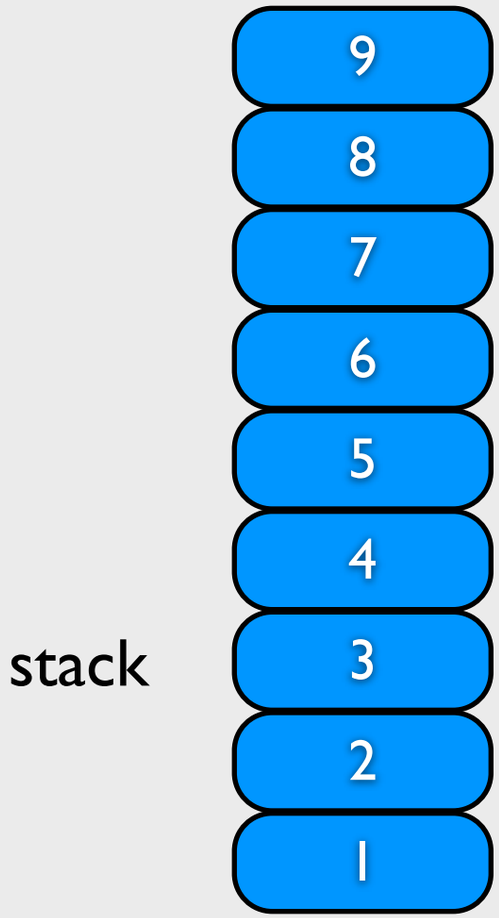
- Example: stack/test.c

```
39      /*************************************************************************
40       *  Initialize the stack.                                                *
41       *************************************************************************/
42      stack_init(&stack, free);
43
44      /*************************************************************************
45       *  Perform some stack operations.                                       *
46       *************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
39  /*****************************************************************************
40   *  Initialize the stack.                                                    *
41   *****************************************************************************/
42  stack_init(&stack, free);
43
44  /*****************************************************************************
45   *  Perform some stack operations.                                           *
46   *****************************************************************************/
47  fprintf(stdout, "Pushing 10 elements\n");
48  for (i = 0; i < 10; i++) {
49      if ((data = (int *)malloc(sizeof(int))) == NULL)
50          return 1;
51      *data = i + 1;
52      if (stack_push(&stack, data) != 0)
53          return 1;
54  }
55  print_stack(&stack);
56
57  fprintf(stdout, "Popping 5 elements\n");
58  for (i = 0; i < 5; i++) {
59      if (stack_pop(&stack, (void **)&data) == 0)
60          free(data);
61      else
62          return 1;
63  }
64  print_stack(&stack);
```



Implementation of Stack

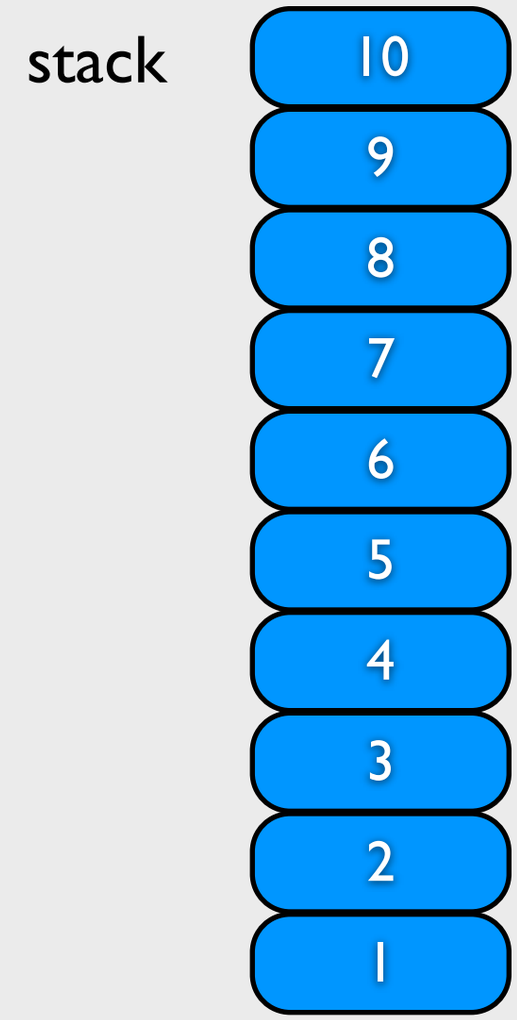
- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

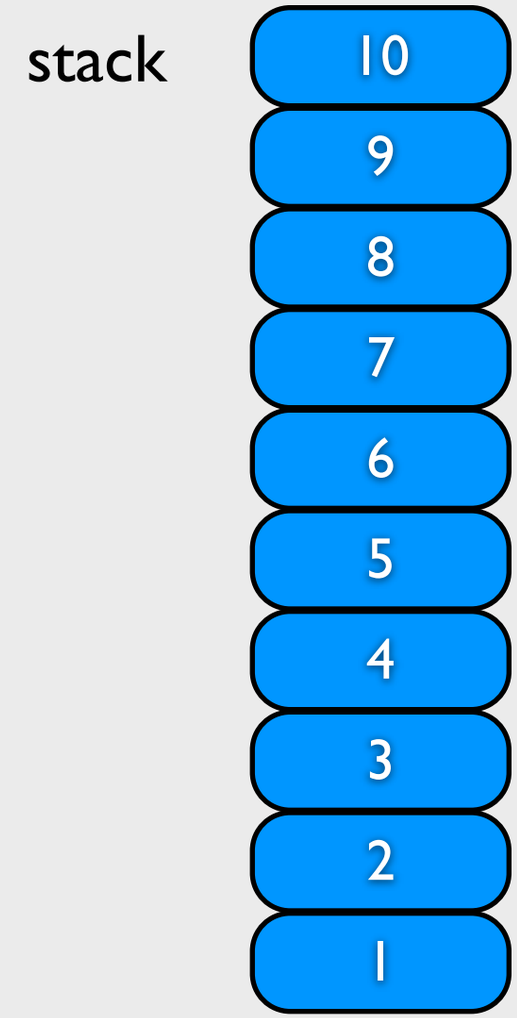
```
39  /*************************************************************************
40   *  Initialize the stack.                                                *
41   *************************************************************************/
42  stack_init(&stack, free);
43
44  /*************************************************************************
45   *  Perform some stack operations.                                       *
46   *************************************************************************/
47  fprintf(stdout, "Pushing 10 elements\n");
48  for (i = 0; i < 10; i++) {
49      if ((data = (int *)malloc(sizeof(int))) == NULL)
50          return 1;
51      *data = i + 1;
52      if (stack_push(&stack, data) != 0)
53          return 1;
54  }
55  print_stack(&stack);
56
57  fprintf(stdout, "Popping 5 elements\n");
58  for (i = 0; i < 5; i++) {
59      if (stack_pop(&stack, (void **)&data) == 0)
60          free(data);
61      else
62          return 1;
63  }
64  print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

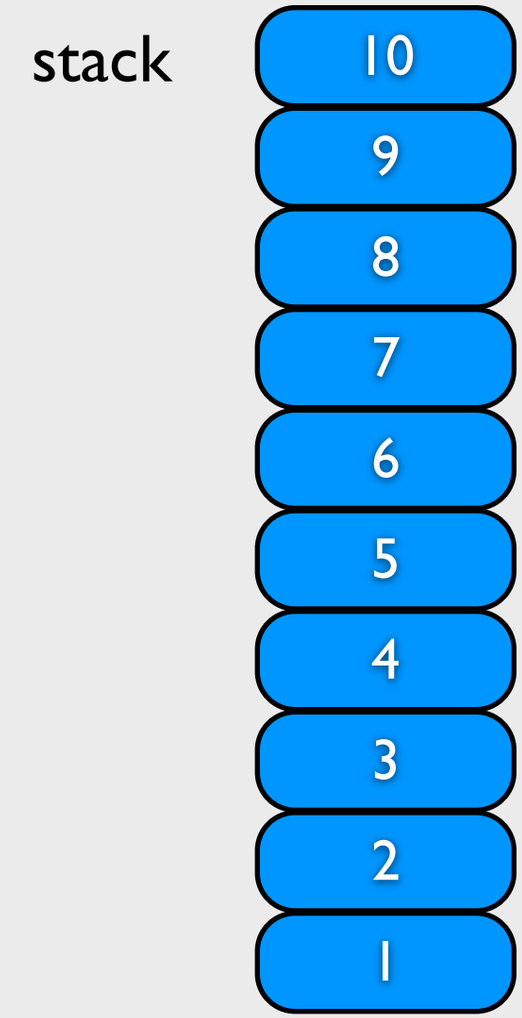
```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

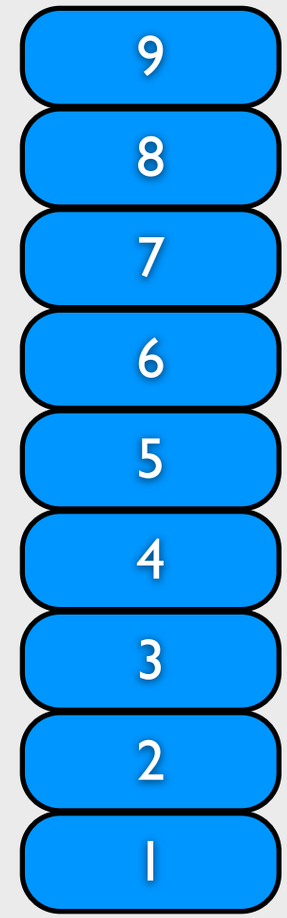
```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

```
39      /*************************************************************************
40       *  Initialize the stack.                                                *
41       *************************************************************************/
42      stack_init(&stack, free);
43
44      /*************************************************************************
45       *  Perform some stack operations.                                       *
46       *************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

```
39  /*************************************************************************
40   *  Initialize the stack.                                                *
41   *************************************************************************/
42  stack_init(&stack, free);
43
44  /*************************************************************************
45   *  Perform some stack operations.                                       *
46   *************************************************************************/
47  fprintf(stdout, "Pushing 10 elements\n");
48  for (i = 0; i < 10; i++) {
49      if ((data = (int *)malloc(sizeof(int))) == NULL)
50          return 1;
51      *data = i + 1;
52      if (stack_push(&stack, data) != 0)
53          return 1;
54  }
55  print_stack(&stack);
56
57  fprintf(stdout, "Popping 5 elements\n");
58  for (i = 0; i < 5; i++) {
59      if (stack_pop(&stack, (void **)&data) == 0)
60          free(data);
61      else
62          return 1;
63  }
64  print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
39  /*****************************************************************************
40   *  Initialize the stack.                                                    *
41   *****************************************************************************/
42  stack_init(&stack, free);
43
44  /*****************************************************************************
45   *  Perform some stack operations.                                           *
46   *****************************************************************************/
47  fprintf(stdout, "Pushing 10 elements\n");
48  for (i = 0; i < 10; i++) {
49      if ((data = (int *)malloc(sizeof(int))) == NULL)
50          return 1;
51      *data = i + 1;
52      if (stack_push(&stack, data) != 0)
53          return 1;
54  }
55  print_stack(&stack);
56
57  fprintf(stdout, "Popping 5 elements\n");
58  for (i = 0; i < 5; i++) {
59      if (stack_pop(&stack, (void **)&data) == 0)
60          free(data);
61      else
62          return 1;
63  }
64  print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

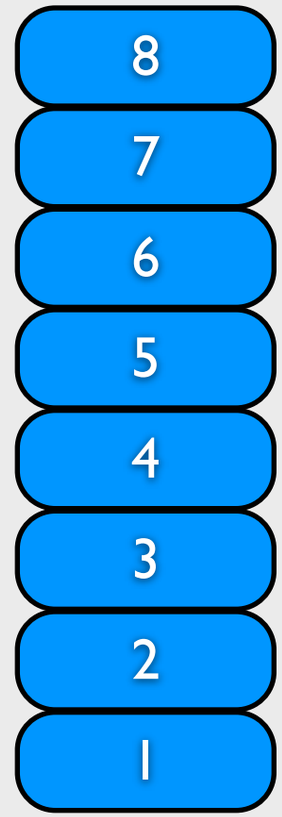
```
39  /*************************************************************************
40   *  Initialize the stack.                                                *
41   *************************************************************************/
42  stack_init(&stack, free);
43
44  /*************************************************************************
45   *  Perform some stack operations.                                       *
46   *************************************************************************/
47  fprintf(stdout, "Pushing 10 elements\n");
48  for (i = 0; i < 10; i++) {
49      if ((data = (int *)malloc(sizeof(int))) == NULL)
50          return 1;
51      *data = i + 1;
52      if (stack_push(&stack, data) != 0)
53          return 1;
54  }
55  print_stack(&stack);
56
57  fprintf(stdout, "Popping 5 elements\n");
58  for (i = 0; i < 5; i++) {
59      if (stack_pop(&stack, (void **)&data) == 0)
60          free(data);
61      else
62          return 1;
63  }
64  print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
39  /*************************************************************************
40   *  Initialize the stack.                                                *
41   *************************************************************************/
42  stack_init(&stack, free);
43
44  /*************************************************************************
45   *  Perform some stack operations.                                       *
46   *************************************************************************/
47  fprintf(stdout, "Pushing 10 elements\n");
48  for (i = 0; i < 10; i++) {
49      if ((data = (int *)malloc(sizeof(int))) == NULL)
50          return 1;
51      *data = i + 1;
52      if (stack_push(&stack, data) != 0)
53          return 1;
54  }
55  print_stack(&stack);
56
57  fprintf(stdout, "Popping 5 elements\n");
58  for (i = 0; i < 5; i++) {
59      if (stack_pop(&stack, (void **)&data) == 0)
60          free(data);
61      else
62          return 1;
63  }
64  print_stack(&stack);
```

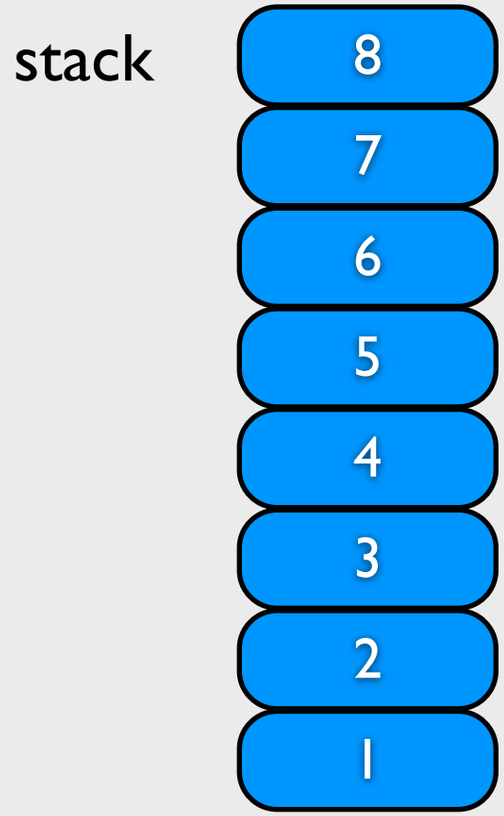
stack



Implementation of Stack

- Example: stack/test.c

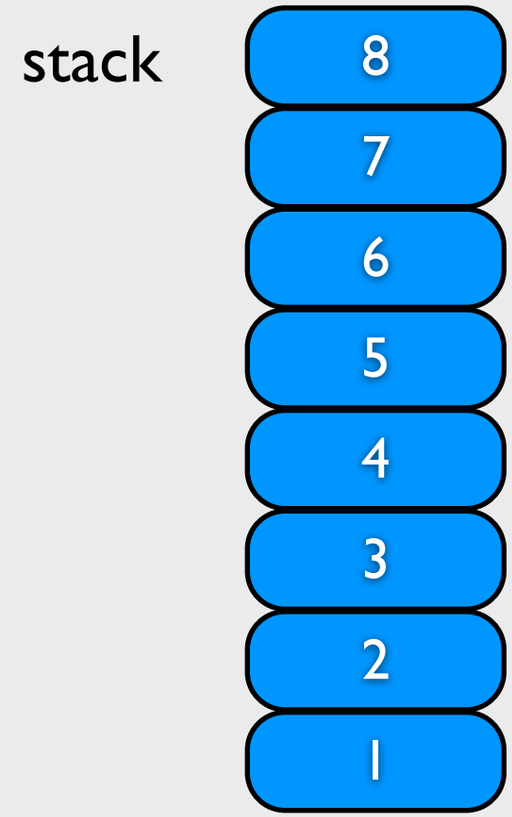
```
39      /*************************************************************************
40       *  Initialize the stack.                                                *
41       *************************************************************************/
42      stack_init(&stack, free);
43
44      /*************************************************************************
45       *  Perform some stack operations.                                       *
46       *************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

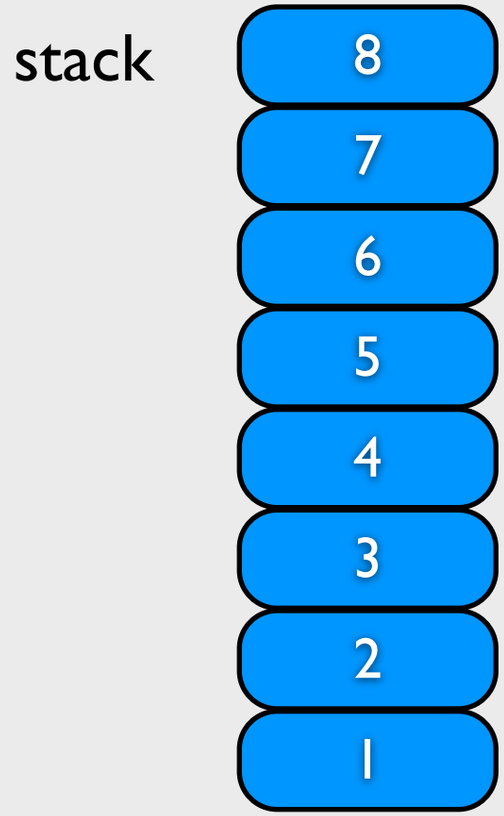
```
39  /*************************************************************************
40   *  Initialize the stack.                                                *
41   *************************************************************************/
42  stack_init(&stack, free);
43
44  /*************************************************************************
45   *  Perform some stack operations.                                       *
46   *************************************************************************/
47  fprintf(stdout, "Pushing 10 elements\n");
48  for (i = 0; i < 10; i++) {
49      if ((data = (int *)malloc(sizeof(int))) == NULL)
50          return 1;
51      *data = i + 1;
52      if (stack_push(&stack, data) != 0)
53          return 1;
54  }
55  print_stack(&stack);
56
57  fprintf(stdout, "Popping 5 elements\n");
58  for (i = 0; i < 5; i++) {
59      if (stack_pop(&stack, (void **)&data) == 0)
60          free(data);
61      else
62          return 1;
63  }
64  print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

```
39      /*************************************************************************
40       *  Initialize the stack.                                                *
41       *************************************************************************/
42      stack_init(&stack, free);
43
44      /*************************************************************************
45       *  Perform some stack operations.                                       *
46       *************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

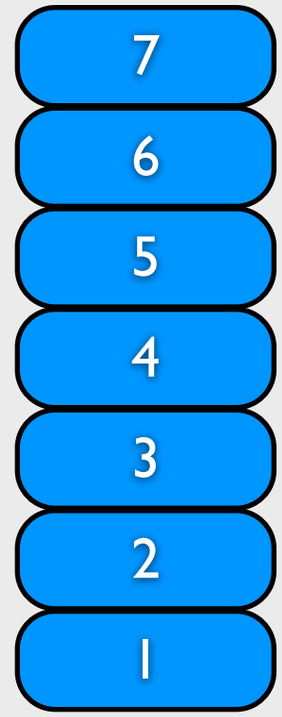


Implementation of Stack

- Example: stack/test.c

```
39      /*********************************************************************
40       *  Initialize the stack.                                            *
41       *********************************************************************/
42      stack_init(&stack, free);
43
44      /*********************************************************************
45       *  Perform some stack operations.                                   *
46       *********************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

stack



Implementation of Stack

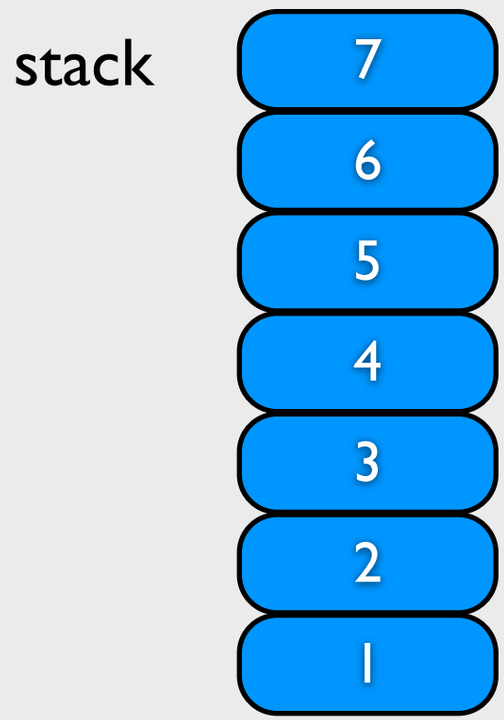
- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

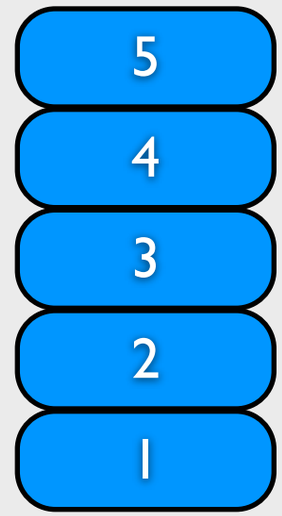
stack

Implementation of Stack

- Example: stack/test.c

```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```

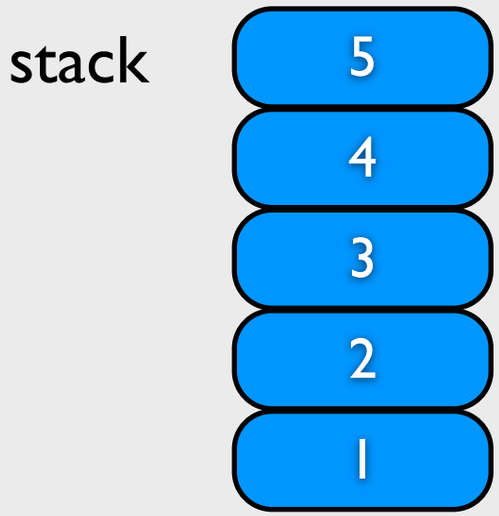
stack



Implementation of Stack

- Example: stack/test.c

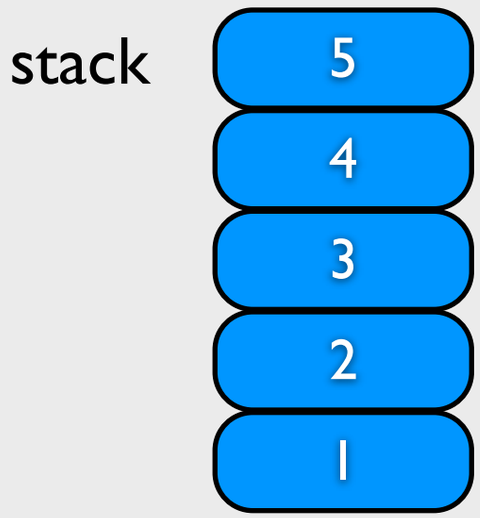
```
39      /*****************************************************************************
40       *  Initialize the stack.                                                    *
41       *****************************************************************************/
42      stack_init(&stack, free);
43
44      /*****************************************************************************
45       *  Perform some stack operations.                                           *
46       *****************************************************************************/
47      fprintf(stdout, "Pushing 10 elements\n");
48      for (i = 0; i < 10; i++) {
49          if ((data = (int *)malloc(sizeof(int))) == NULL)
50              return 1;
51          *data = i + 1;
52          if (stack_push(&stack, data) != 0)
53              return 1;
54      }
55      print_stack(&stack);
56
57      fprintf(stdout, "Popping 5 elements\n");
58      for (i = 0; i < 5; i++) {
59          if (stack_pop(&stack, (void **)&data) == 0)
60              free(data);
61          else
62              return 1;
63      }
64      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

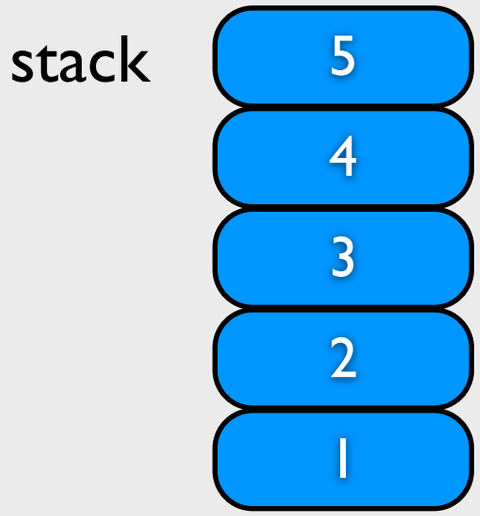
```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

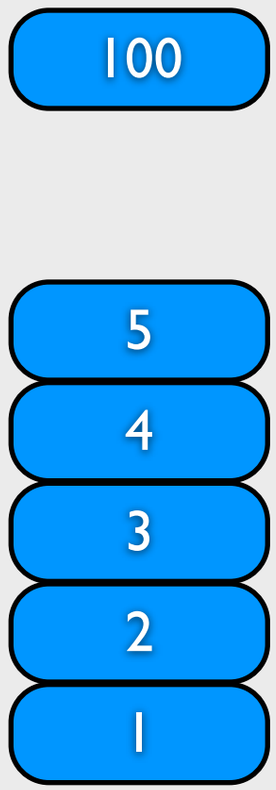
```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```



Implementation of Stack

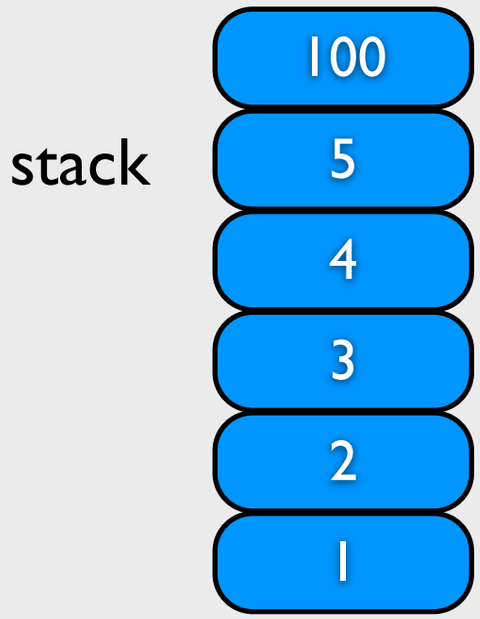
- Example: stack/test.c

```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```

Implementation of Stack

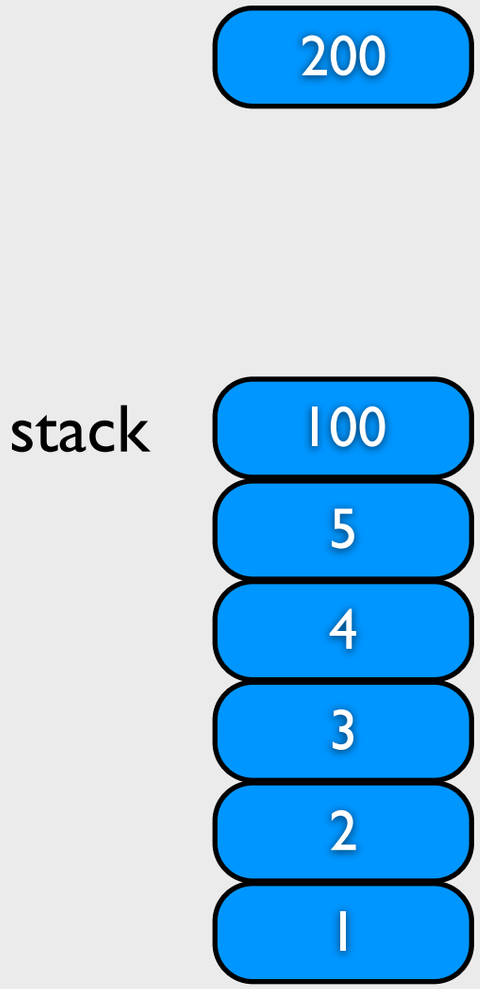
- Example: stack/test.c

```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

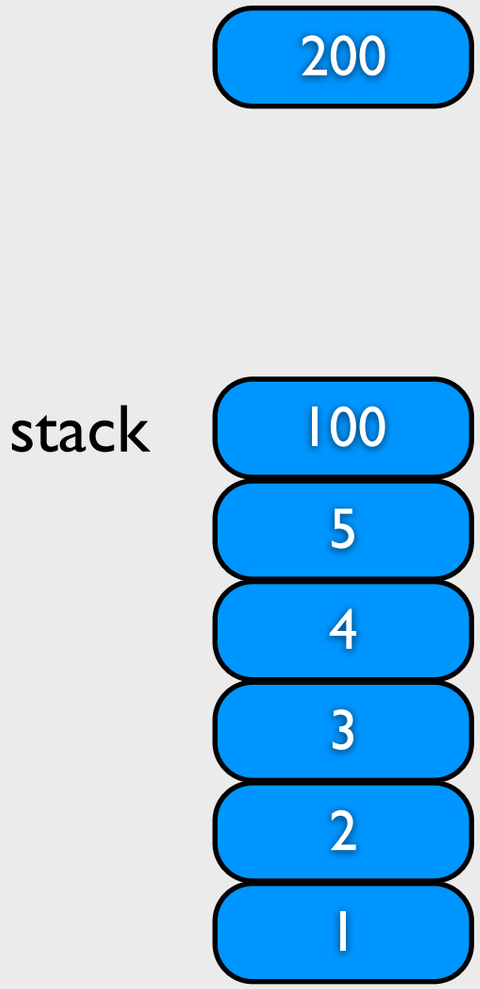
```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

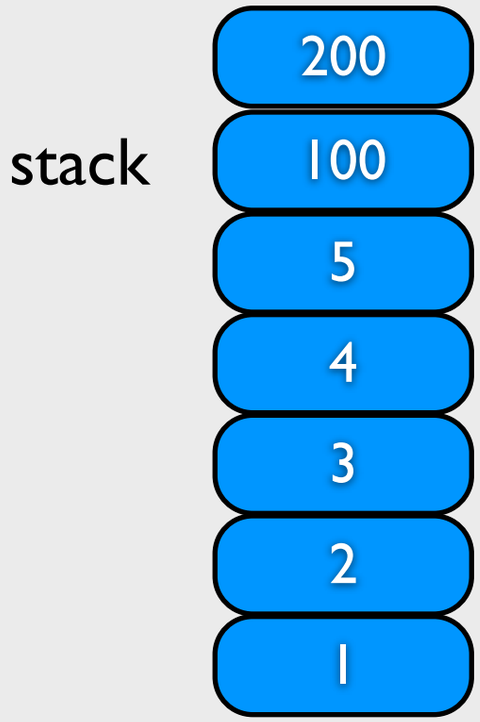
```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```



Implementation of Stack

- Example: stack/test.c

```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
66      fprintf(stdout, "Pushing 100 and 200\n");
67      if ((data = (int *)malloc(sizeof(int))) == NULL)
68          return 1;
69      *data = 100;
70      if (stack_push(&stack, data) != 0)
71          return 1;
72
73      if ((data = (int *)malloc(sizeof(int))) == NULL)
74          return 1;
75      *data = 200;
76      if (stack_push(&stack, data) != 0)
77          return 1;
78      print_stack(&stack);
```

Implementation of Stack

- Example: stack/test.c

```
80      if ((data = stack_peek(&stack)) != NULL)
81          fprintf(stdout, "Peeking at the top element...Value=%03d\n", *data);
82      else
83          fprintf(stdout, "Peeking at the top element...Value=NULL\n");
84      print_stack(&stack);
85
86      fprintf(stdout, "Popping all elements\n");
87      while (stack_size(&stack) > 0) {
88          if (stack_pop(&stack, (void **)&data) == 0)
89              free(data);
90      }
```

Implementation of Stack

- Example: stack/test.c

```
80      if ((data = stack_peek(&stack)) != NULL)
81          fprintf(stdout, "Peeking at the top element...Value=%03d\n", *data);
82      else
83          fprintf(stdout, "Peeking at the top element...Value=NULL\n");
84      print_stack(&stack);
85
86      fprintf(stdout, "Popping all elements\n");
87      while (stack_size(&stack) > 0) {
88          if (stack_pop(&stack, (void **)&data) == 0)
89              free(data);
90      }
```

Implementation of Stack

- Example: stack/test.c

```
80      if ((data = stack_peek(&stack)) != NULL)
81          fprintf(stdout, "Peeking at the top element...Value=%03d\n", *data);
82      else
83          fprintf(stdout, "Peeking at the top element...Value=NULL\n");
84      print_stack(&stack);
85
86      fprintf(stdout, "Popping all elements\n");
87      while (stack_size(&stack) > 0) {
88          if (stack_pop(&stack, (void **)&data) == 0)
89              free(data);
90      }
```

Implementation of Stack

- Example: stack/test.c

```
80      if ((data = stack_peek(&stack)) != NULL)
81          fprintf(stdout, "Peeking at the top element...Value=%03d\n", *data);
82      else
83          fprintf(stdout, "Peeking at the top element...Value=NULL\n");
84      print_stack(&stack);
85
86      fprintf(stdout, "Popping all elements\n");
87      while (stack_size(&stack) > 0) {
88          if (stack_pop(&stack, (void **)&data) == 0)
89              free(data);
90      }
```

Implementation of Stack

- Example: stack/test.c

```
80      if ((data = stack_peek(&stack)) != NULL)
81          fprintf(stdout, "Peeking at the top element...Value=%03d\n", *data);
82      else
83          fprintf(stdout, "Peeking at the top element...Value=NULL\n");
84      print_stack(&stack);
85
86      fprintf(stdout, "Popping all elements\n");
87      while (stack_size(&stack) > 0) {
88          if (stack_pop(&stack, (void **)&data) == 0)
89              free(data);
90      }
```

Implementation of Stack

- Example: stack/test.c

```
80      if ((data = stack_peek(&stack)) != NULL)
81          fprintf(stdout, "Peeking at the top element...Value=%03d\n", *data);
82      else
83          fprintf(stdout, "Peeking at the top element...Value=NULL\n");
84      print_stack(&stack);
85
86      fprintf(stdout, "Popping all elements\n");
87      while (stack_size(&stack) > 0) {
88          if (stack_pop(&stack, (void **)&data) == 0)
89              free(data);
90      }
```

Implementation of Stack

- Example: stack/test.c

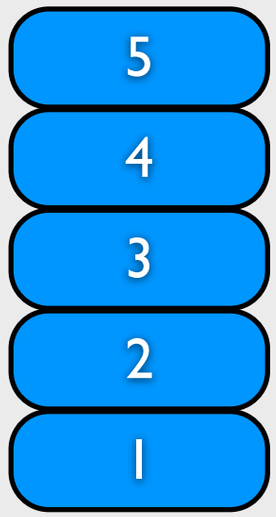
```
80      if ((data = stack_peek(&stack)) != NULL)
81          fprintf(stdout, "Peeking at the top element...Value=%03d\n", *data);
82      else
83          fprintf(stdout, "Peeking at the top element...Value=NULL\n");
84      print_stack(&stack);
85
86      fprintf(stdout, "Popping all elements\n");
87      while (stack_size(&stack) > 0) {
88          if (stack_pop(&stack, (void **)&data) == 0)
89              free(data);
90      }
```

Implementation of Stack

- Example: stack/test.c

```
80      if ((data = stack_peek(&stack)) != NULL)
81          fprintf(stdout, "Peeking at the top element...Value=%03d\n", *data);
82      else
83          fprintf(stdout, "Peeking at the top element...Value=NULL\n");
84      print_stack(&stack);
85
86      fprintf(stdout, "Popping all elements\n");
87      while (stack_size(&stack) > 0) {
88          if (stack_pop(&stack, (void **)&data) == 0)
89              free(data);
90      }
```

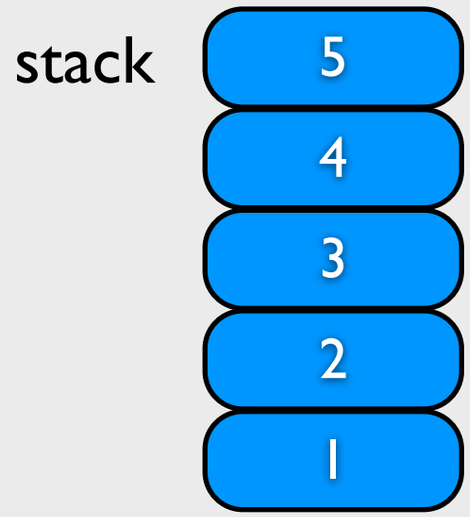
stack



Implementation of Stack

- Example: stack/test.c

```
80      if ((data = stack_peek(&stack)) != NULL)
81          fprintf(stdout, "Peeking at the top element...Value=%03d\n", *data);
82      else
83          fprintf(stdout, "Peeking at the top element...Value=NULL\n");
84      print_stack(&stack);
85
86      fprintf(stdout, "Popping all elements\n");
87      while (stack_size(&stack) > 0) {
88          if (stack_pop(&stack, (void **)&data) == 0)
89              free(data);
90      }
```



Implementation of Stack

- Example: stack/test.c

```
 99    /*********************************************************************************
100     *  Destroy the stack.                                                           *
101     *********************************************************************************/
102    fprintf(stdout, "Destroying the stack\n");
103    stack_destroy(&stack);
104
```

Implementation of Stack

- Example: stack/test.c

```
 99    /*****************************************************************************
100     *  Destroy the stack.                                                       *
101     *****************************************************************************/
102    fprintf(stdout, "Destroying the stack\n");
103    stack_destroy(&stack);
104
```