# Intermediate Programming

Day 16

## Outline

- Exercise 5-3
- Linked lists
- Review questions

## Exercise 5-3 (part 2)

Add a definition of `uniform_rand` … should return pseudo-random integers in the range `0` to `NUM_BUCKETS-1`, inclusive.

```
int uniform_rand( void ){ return rand() % NUM_BUCKETS; }
```

## Exercise 5-3 (part 2)

Add code to generate 500 uniformly-generated pseudo-random integers and increment the elements of the `hist` array accordingly.

```
for( int i=0 ; i<500 ; i++ ) hist[ normal_rand() ]++;
```

## Exercise 5-3 (part 2)

Add a definition for the `print_hist` function… It should print a bar graph of the contents of its array.

```
void print_hist( int counts[] )
{
    for( int i=0 ; i<NUM_BUCKETS ; i++ )
    {
            printf( "%d: " , i );
            for( int j=0 ; j<counts[i] ; j++ ) printf( "*" );
            printf( "\n" );
    }
}
```

## Exercise 5-3 (part 2)

Add a definition for the normal_rand function… This function should return a normal distribution of integers... in the range 0 to NUM_BUCKETS-1.

```
int normal_rand( void )
{
    static const int COUNT = 6;
    int sum = 0;
    for( int i=0 ; i<COUNT ; i++ ) sum += uniform_rand();
    return sum/COUNT;
}
```

# Exercise 5-3 (part 3)

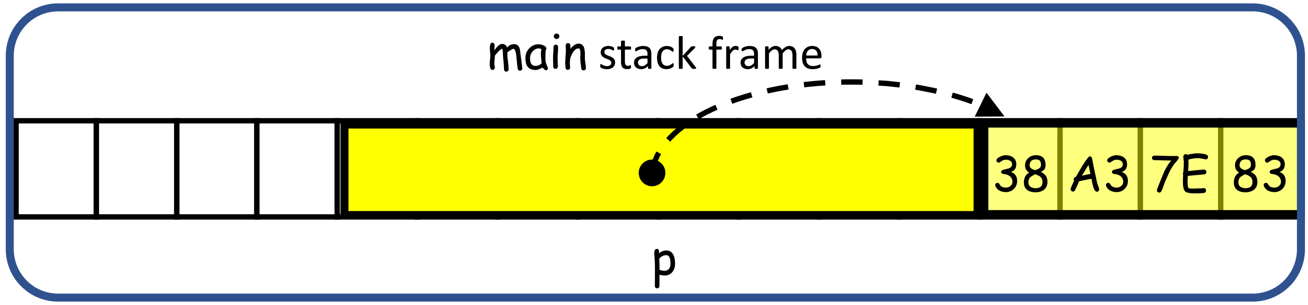
Implement the int_magnitude function… It should return an unsigned int value representing the *magnitude* of the argument value.

```
unsigned int magnitude( unsigned int value )
{
    unsigned int mask = 1<<31;
    if( value & mask ) return ~value + 1;
    else               return value;
}
```

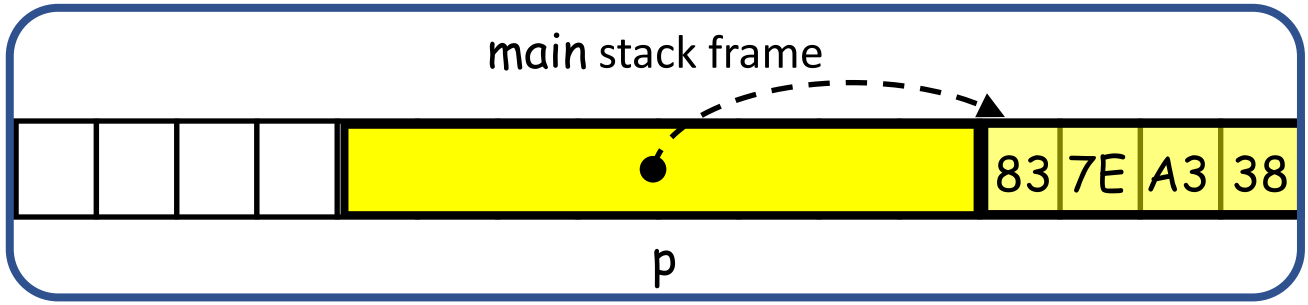
```c
#include <stdio.h>
int main( void )
{
        ...
        unsigned int val = 950238851u;
        unsigned int *p = &val;
        printf( "%u\n" , *p );
        return 0;
}
```

# Exercise 5-3 (part 4)

val ... in hexadecimal is 38A37E83... in gdb ... print each byte ... is the computer you are running on ... big endian or little endian?



or

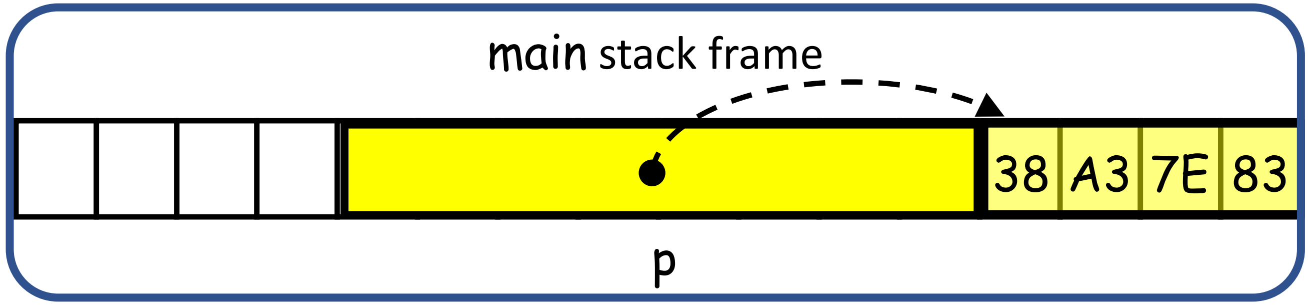


```
#include <stdio.h>
int main( void )
{
        ...
        unsigned int val = 950238851u;
        unsigned int *p = &val;
        printf( "%u\n" , *p );
        return 0;
}
```

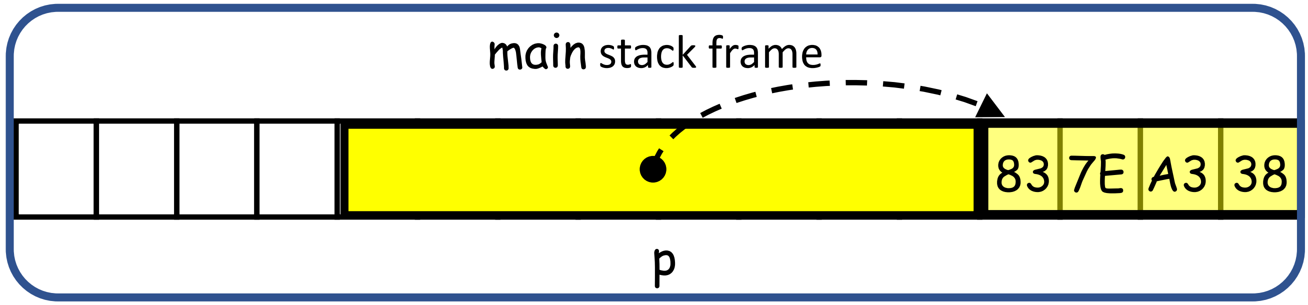
# Exercise 5-3 (part 4)

val … in hexadecimal is 38A37E83… in gdb … print each byte … is the computer you are running on … big endian or little endian?

```
    (gdb) b main
    ...
    (gdb) r
    ...
    (gdb) n
    ...
    (gdb) n
    21          printf("%u\n", *p);
```



or

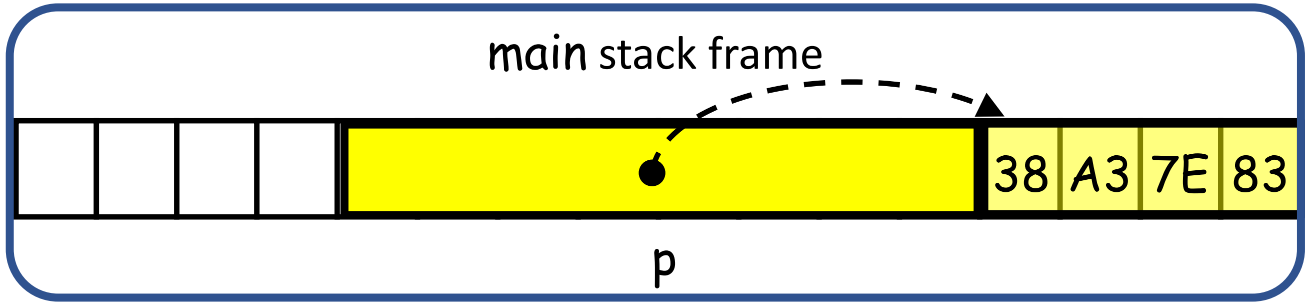


## Exercise 5-3 (part 4)

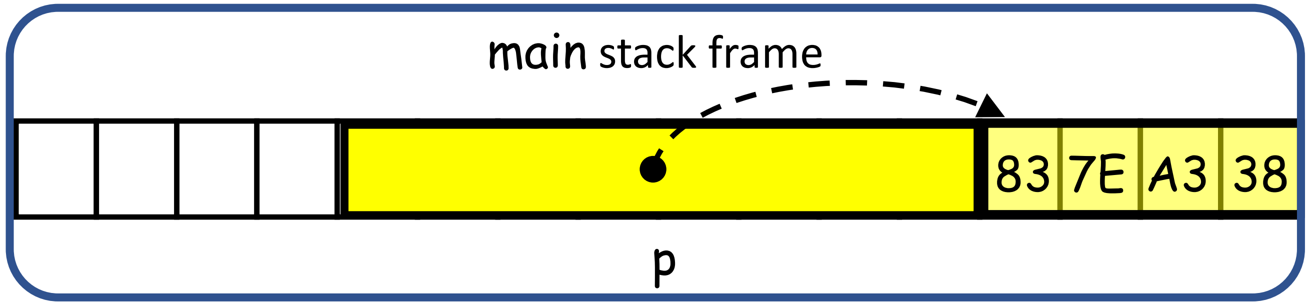
```
#include <stdio.h>
int main( void )
{
    ...
    unsigned int val = 950238851u;
    unsigned int *p = &val;
    printf( "%u\n" , *p );
    return 0;
}
```

val … in hexadecimal is 38A37E83… in gdb … print each byte … is the computer you are running on … big endian or little endian?

```
(gdb) p /x ((unsigned char*)p)[0]
$1 = 0x83
(gdb) p /x ((unsigned char*)p)[1]
$2 = 0x7e
(gdb) p /x ((unsigned char*)p)[2]
$3 = 0xa3
(gdb) p /x ((unsigned char*)p)[3]
$4 = 0x38
```



or



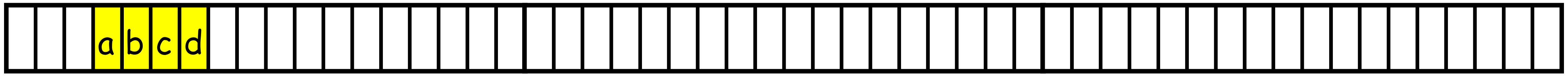
# Outline

- Exercise 5-3
- Linked lists
- Review questions

# Linked lists

- Arrays:
  - ✓ Contiguous memory
    ⇒ Fast (constant time) look-up
  - ✗ Do not support dynamic insertion/deletion

```
…
char ar[] = { 'a' , 'b' , 'c' , 'd' };
…
```

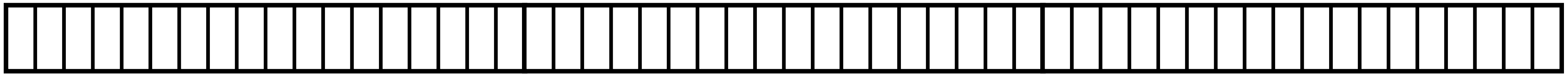


*address space*

## Linked lists

- Arrays:
  - ✓Contiguous memory
    ⇒ Fast (constant time) look-up
  - ✘Do not support dynamic insertion/deletion
- Linked lists:
  - ✓Support dynamic insertion/deletion
  - ✘Discontiguous memory
    ⇒ Slow (linear time) look-up
  - ✘Explicit pointer storage

```
…
typedef struct _Node
{
    struct _Node *next;
    char value;
} Node;






…
```

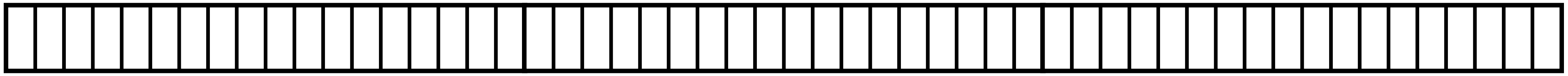

*address space*

```
…
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;




…
```

# Linked lists

- Arrays:
  - ✓Contiguous memory
    ⇒ Fast (constant time) look-up
  - ✘Do not support dynamic insertion/deletion
- Li
  - support dynamic insertion/deletion
  - ✘Discontiguous memory
    ⇒ Slow (linear time) look-up
  - ✘Explicit pointer storage

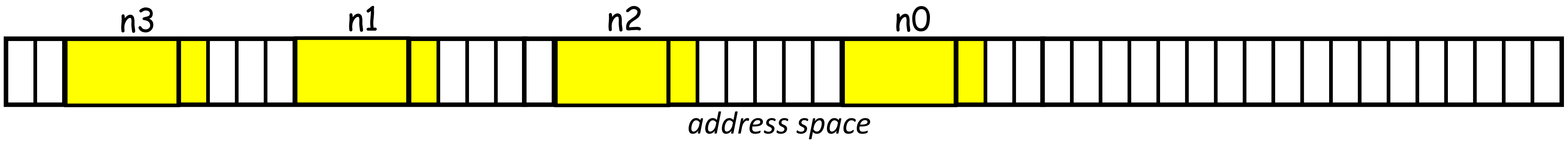
Note that the `struct` cannot be unnamed since we need to access it within the `struct`, before the `typedef` is complete.


*address space*

# Linked lists

- Arrays:
  - ✓Contiguous memory
    ⇒ Fast (constant time) look-up
  - ✘Do not support dynamic insertion/deletion
- Linked lists:
  - ✓Support dynamic insertion/deletion
  - ✘Discontiguous memory
    ⇒ Slow (linear time) look-up
  - ✘Explicit pointer storage

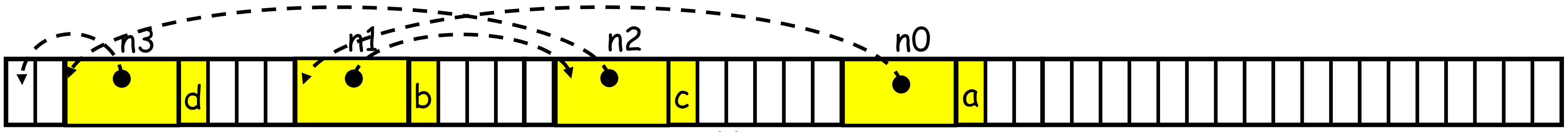
```
…
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *n0 = malloc( sizeof( Node ) );
Node *n1 = malloc( sizeof( Node ) );
Node *n2 = malloc( sizeof( Node ) );
Node *n3 = malloc( sizeof( Node ) );




…
```



# Linked lists

- Arrays:
  - ✓Contiguous memory
    ⇒ Fast (constant time) look-up
  - ✘ Do not support dynamic insertion/deletion
- Linked lists:
  - ✓Support dynamic insertion/deletion
  - ✘ Discontiguous memory
    ⇒ Slow (linear time) look-up
  - ✘ Explicit pointer storage

```
…
typedef struct _Node
{
      struct _Node *next;
      char value;
} Node;

Node *n0 = malloc( sizeof( Node ) );
Node *n1 = malloc( sizeof( Node ) );
Node *n2 = malloc( sizeof( Node ) );
Node *n3 = malloc( sizeof( Node ) );

n0->value = 'a' ; n0->next = le1;
n1->value = 'b' ; n1->next = le2;
n2->value = 'c' ; n2->next = le3;
n3->value = 'd' ; n3->next = NULL;
…
```



*address space*

## Linked lists

- Basic operations:
  - Create a node
  - Add a node
  - …
- Terminology:
  - The first element of a linked list is the "head"

*charList.h*

```
typedef struct _Node
{
    struct _Node *next;
    char value;
} Node;


…
```

## Linked lists

- Create a node
  - Allocate the linked-list element
  - Set its members

*charList.h*

```
typedef struct _Node
{
    struct _Node *next;
    char value;
} Node;

Node *create_node( char c );
…
```

*charList.c*

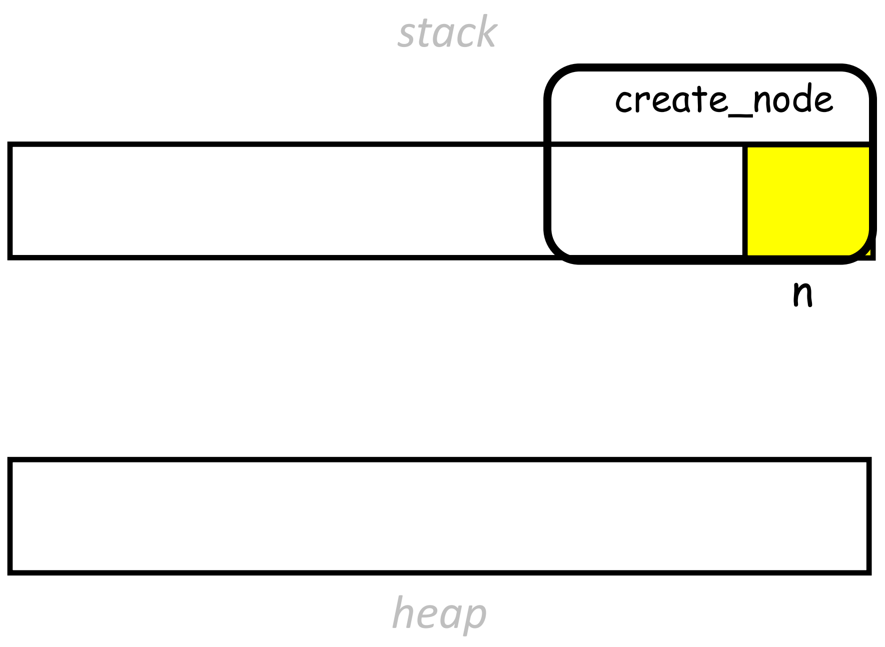
```
#include "charList.h"
#include <stdlib.h>

Node *create_node( char c )
{
    Node *n = malloc( sizeof( Node ) );
    if( !n ) return NULL;
    n->next = NULL ; n->value = c;
    return n;
}
```

## Linked lists

- Create a node
  - Allocate the linked-list element
  - Set its members

```
charList.h
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
…
```

```
charList.c
#include "charList.h"
#include <stdlib.h>

Node *create_node( char c )
{
	Node *n = malloc( sizeof( Node ) );
	if( !n ) return NULL;
	n->next = NULL ; n->value = c;
	return n;
}
```

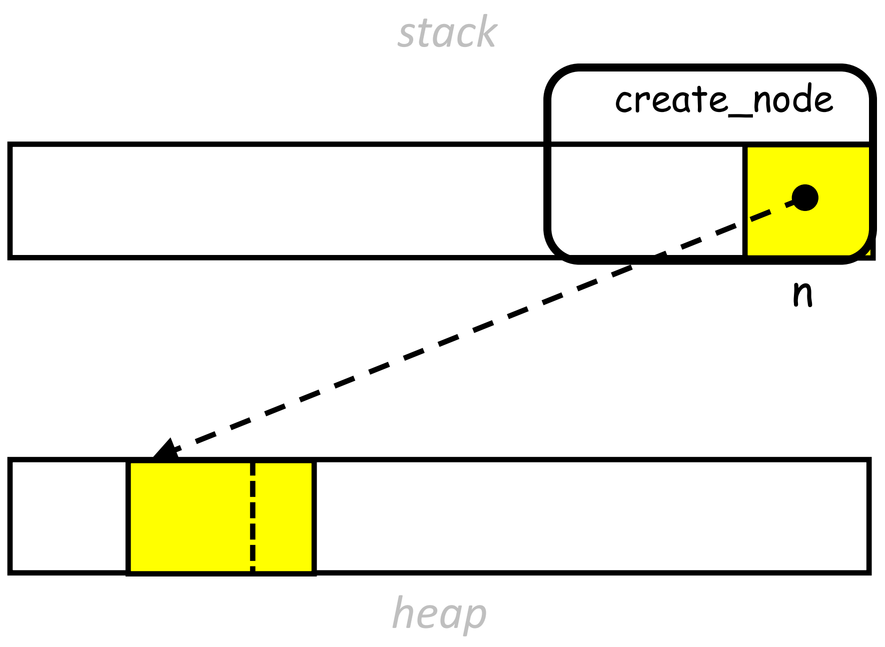


# Linked lists

- Create a node
  - Allocate the linked-list element
  - Set its members

charList.h
```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
…
```

charList.c
```
#include "charList.h"
#include <stdlib.h>

Node *create_node( char c )
{
	Node *n = malloc( sizeof( Node ) );
	if( !n ) return NULL;
	n->next = NULL ; n->value = c;
	return n;
}
```

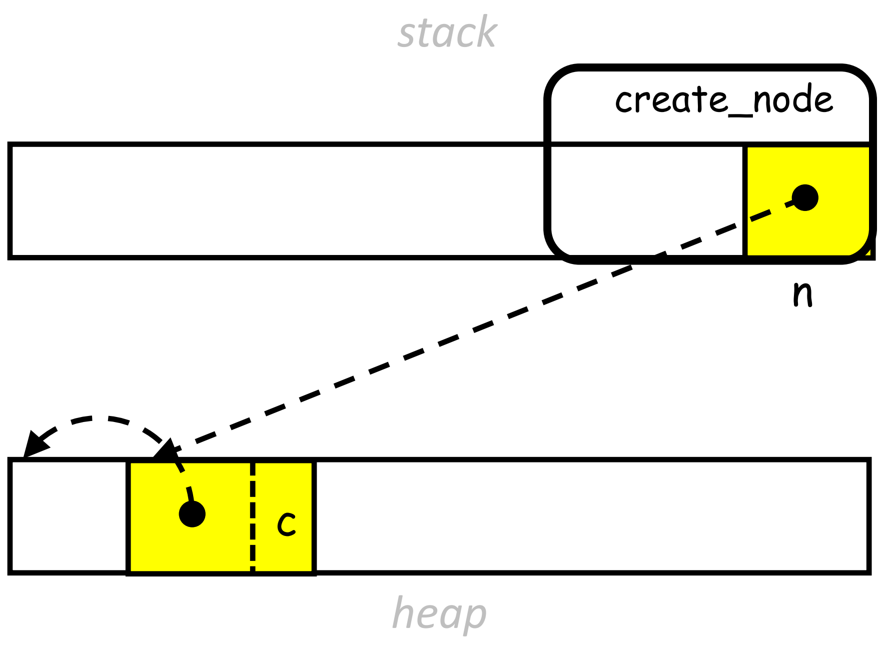


# Linked lists

- Create a node
  - Allocate the linked-list element
  - Set its members

*charList.h*
```
typedef struct _Node
{
    struct _Node *next;
    char value;
} Node;

Node *create_node( char c );
…
```

*charList.c*
```
#include "charList.h"
#include <stdlib.h>

Node *create_node( char c )
{
    Node *n = malloc( sizeof( Node ) );
    if( !n ) return NULL;
    n->next = NULL ; n->value = c;
    return n;
}
```

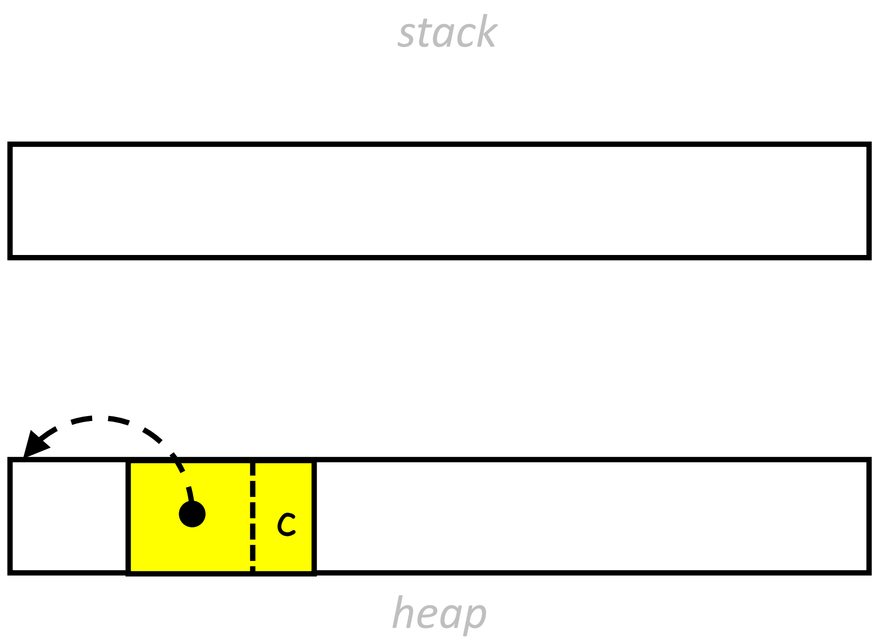


## Linked lists

- Create a node
  - Allocate the linked-list element
  - Set its members

charList.h
```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
…
```

charList.c
```
#include "charList.h"
#include <stdlib.h>

Node *create_node( char c )
{
	Node *n = malloc( sizeof( Node ) );
	if( !n ) return NULL;
	n->next = NULL ; n->value = c;
	return n;
}
```

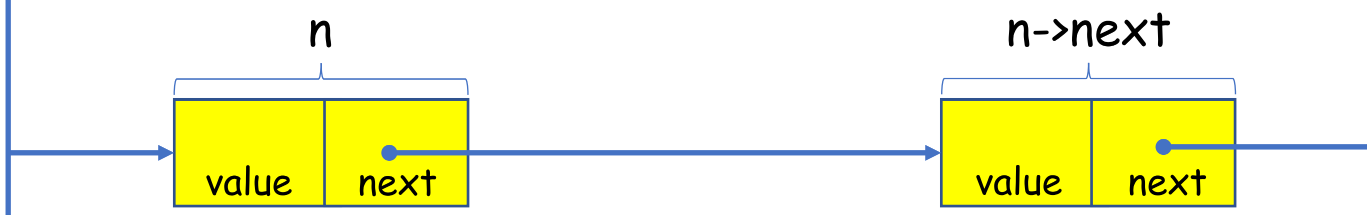


## Linked lists

- Add a node
  - Create the node
  - Update the pointers

charList.h
```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
…
```

charList.c
```
#include "charList.h"
#include <stdlib.h>
…
int add_after( Node *n , char c )
{
	Node *newN = create_node( c );
	if( !newN ) return 1;
	newN->next = n->next;
	n->next = newN;
	return 0;
}
```



# Linked lists

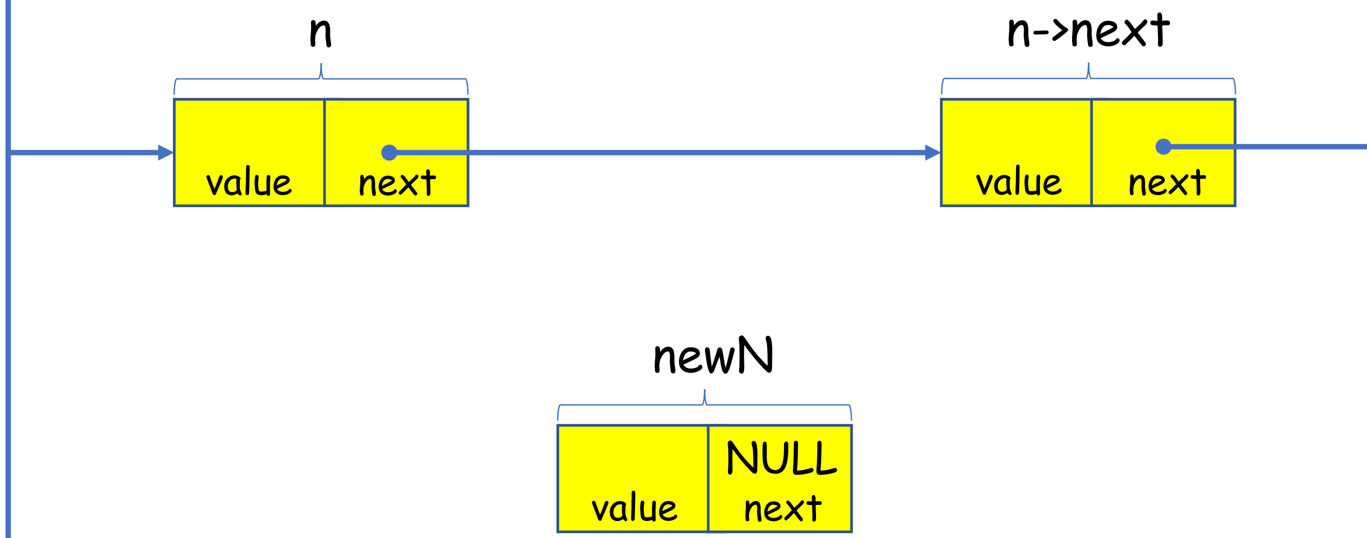
- Add a node
    - **Create the node**
    - Update the pointers

*charList.h*

```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
…
```

*charList.c*

```
#include "charList.h"
#include <stdlib.h>
…
int add_after( Node *n , char c )
{
	Node *newN = create_node( c );
	if( !newN ) return 1;
	newN->next = n->next;
	n->next = newN;
	return 0;
}
```



# Linked lists

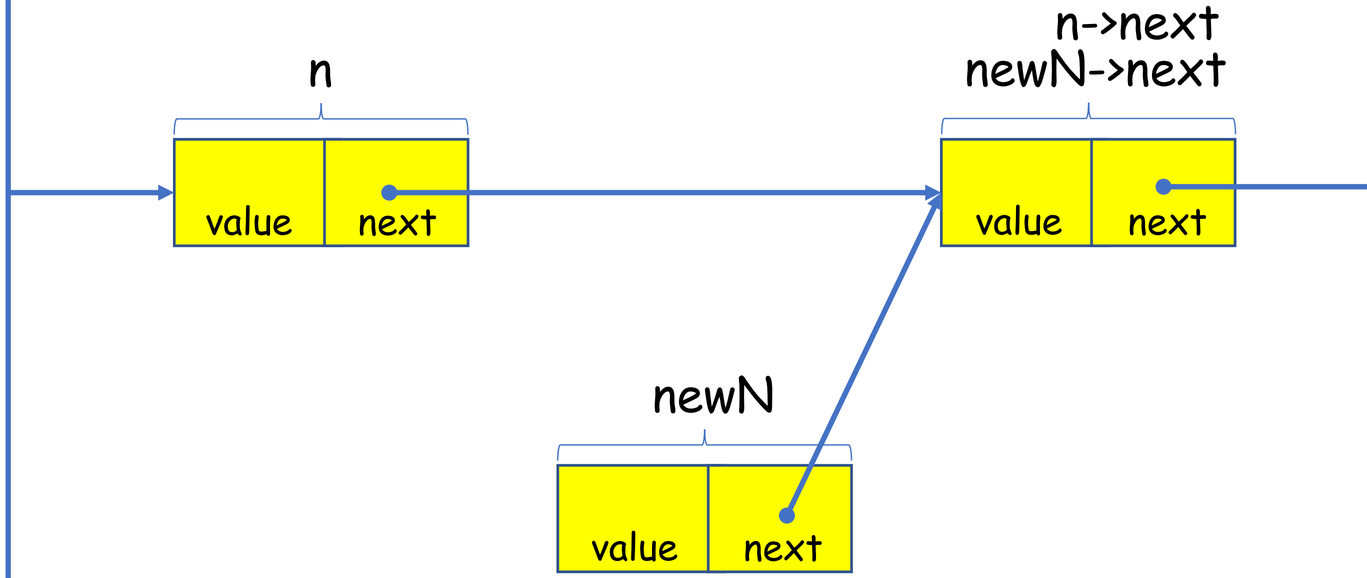
- Add a node
  - Create the node
  - **Update the pointers**

*charList.h*

```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
…
```

*charList.c*

```
#include "charList.h"
#include <stdlib.h>
…
int add_after( Node *n , char c )
{
	Node *newN = create_node( c );
	if( !newN ) return 1;
	newN->next = n->next;
	n->next = newN;
	return 0;
}
```



# Linked lists

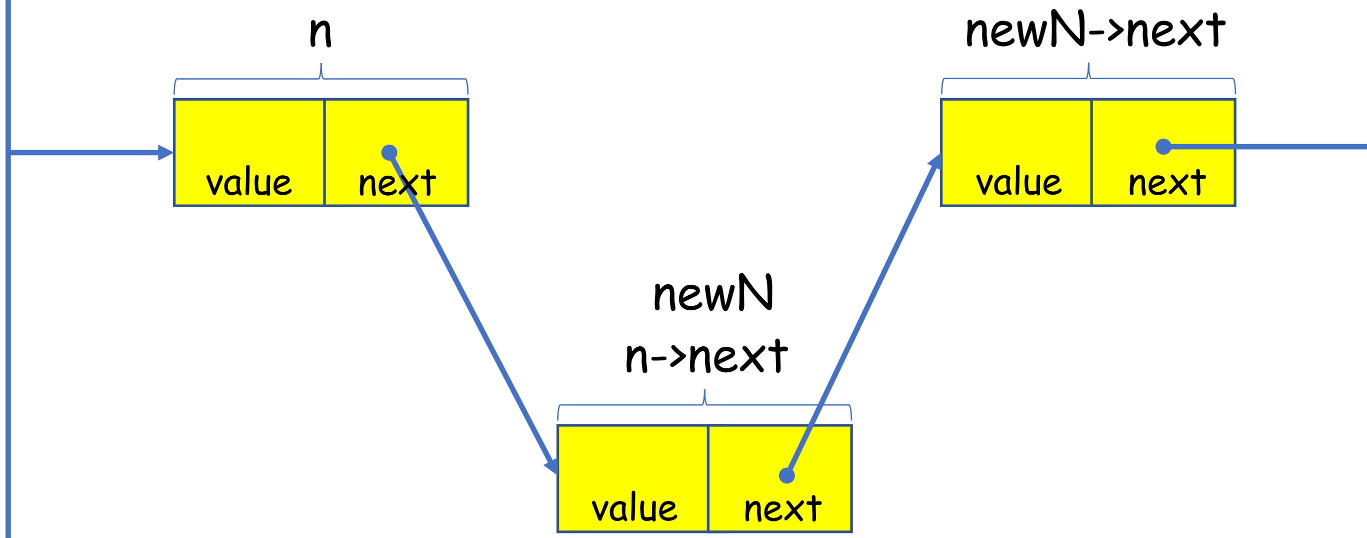
- Add a node
  - Create the node
  - **Update the pointers**

*charList.c*

```
#include "charList.h"
#include <stdlib.h>
…
int add_after( Node *n , char c )
{
	Node *newN = create_node( c );
	if( !newN ) return 1;
	newN->next = n->next;
	n->next = newN;
	return 0;
}
```

*charList.h*

```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
…
```



# Linked lists

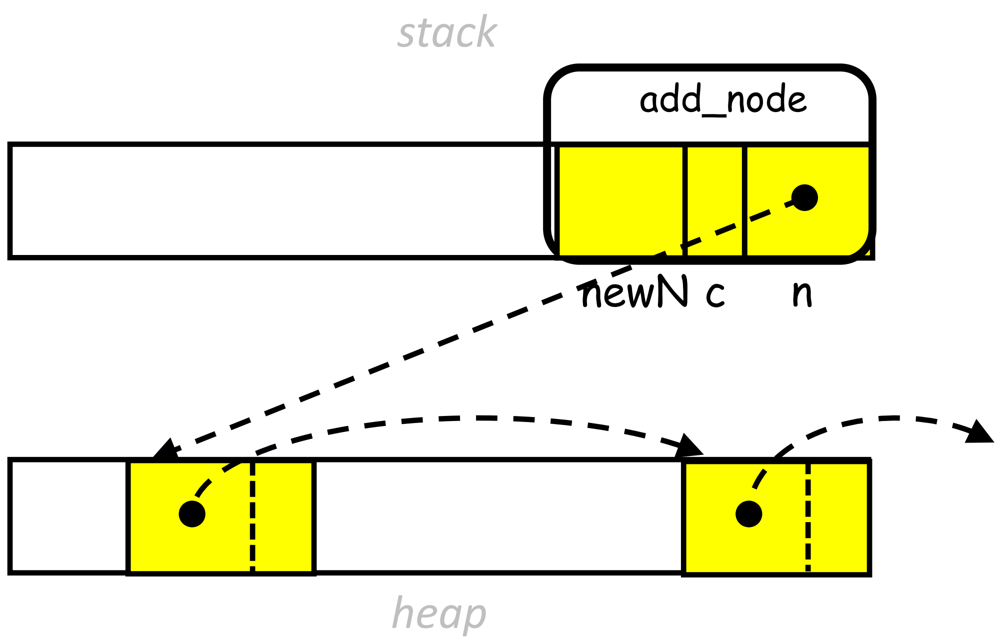
- Add a node
  - Create the node
  - Update the pointers

charList.c

```c
#include "charList.h"
#include <stdlib.h>
…
int add_after( Node *n , char c )
{
    Node *newN = create_node( c );
    if( !newN ) return 1;
    newN->next = n->next;
    n->next = newN;
    return 0;
}
```

charList.h

```c
typedef struct _Node
{
    struct _Node *next;
    char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
…
```



## Linked lists

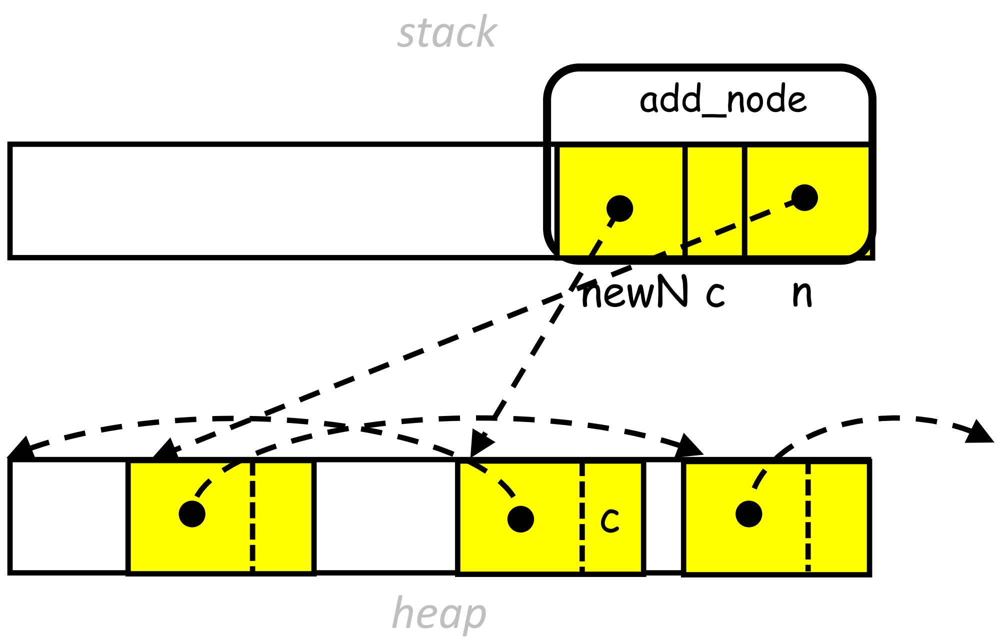
- Add a node
  - Create the node
  - Update the pointers

*charList.h*

```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
…
```

*charList.c*

```
#include "charList.h"
#include <stdlib.h>
…
int add_after( Node *n , char c )
{
	Node *newN = create_node( c );
	if( !newN ) return 1;
	newN->next = n->next;
	n->next = newN;
	return 0;
}
```



# Linked lists

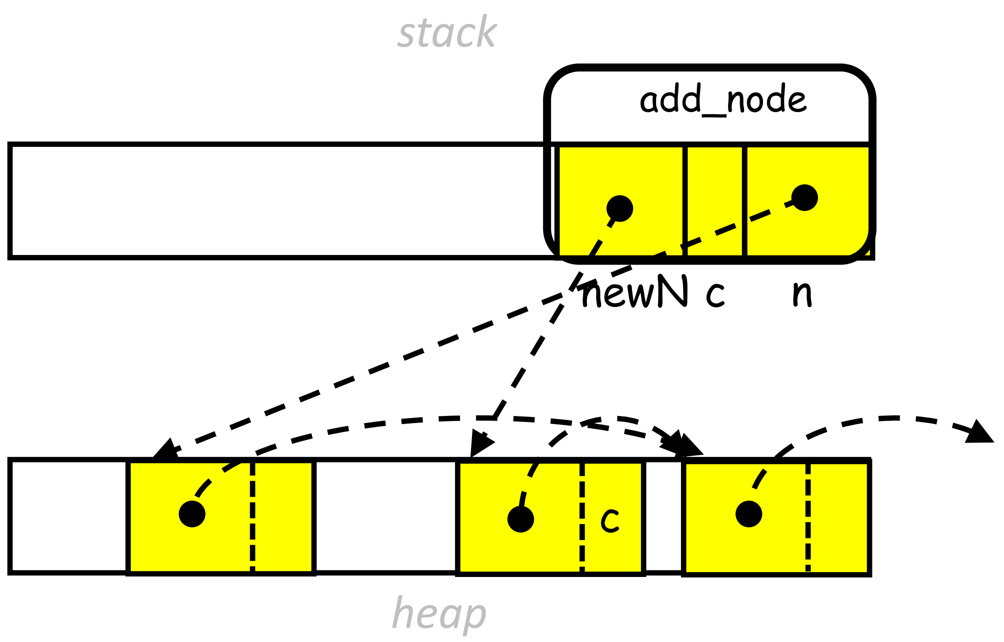
- Add a node
  - Create the node
  - Update the pointers

*charList.c*

```
#include "charList.h"
#include <stdlib.h>
…
int add_after( Node *n , char c )
{
    Node *newN = create_node( c );
    if( !newN ) return 1;
    newN->next = n->next;
    n->next = newN;
    return 0;
}
```

*charList.h*

```
typedef struct _Node
{
    struct _Node *next;
    char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
…
```



# Linked lists

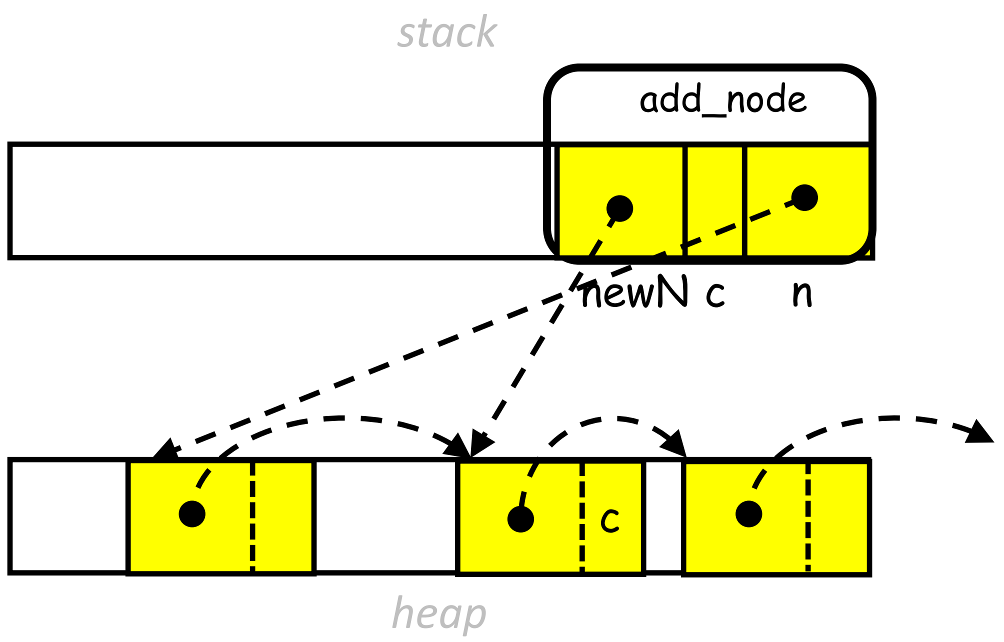
- Add a node
  - Create the node
  - Update the pointers

*charList.c*

```
#include "charList.h"
#include <stdlib.h>
…
int add_after( Node *n , char c )
{
     Node *newN = create_node( c );
     if( !newN ) return 1;
     newN->next = n->next;
     n->next = newN;
     return 0;
}
```

*charList.h*

```
typedef struct _Node
{
     struct _Node *next;
     char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
…
```



## Linked lists

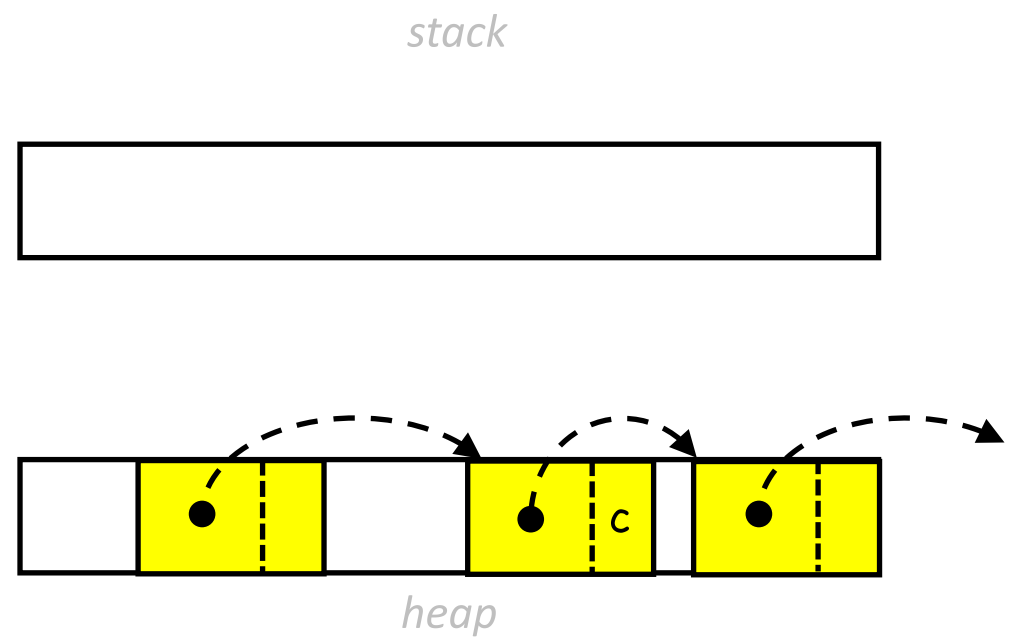
- Add a node
  - Create the node
  - Update the pointers

*charList.h*

```
typedef struct _Node
{
    struct _Node *next;
    char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
…
```

*charList.c*

```
#include "charList.h"
#include <stdlib.h>
…
int add_after( Node *n , char c )
{
    Elem *newN = create_node( c );
    if( !newN ) return 1;
    newN->next = n->next;
    n->next = newN;
    return 0;
}
```



# Linked lists

- Getting the length
  - Increment a counter
  - Advance to the next node (if it isn't NULL)

*charList.h*

```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
int length( const Node *head );
```

*charList.c*

```
#include "charList.h"
#include <stdlib.h>
…
int length( const Node *head )
{
	int len=0;
	while( head ){ len++ ; head = head->next; }
	return len;
}
```

# Linked lists

- Printing
  - Print out the value in the current node
  - Advance to the next node
    (if it isn't NULL)

*charList.h*

```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
int length( const Node *head );
```

*charListIO.h*

```
#include "charList.h"
void print( const Node *head );
```

*charListIO.c*

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

void print( const Node *head )
{
	for( const Node *n=head ; n!=NULL ; n=n->next )
		printf( " %c" , n->value );
	printf( "\n" );
}
```

*main.c*

```
#include <stdio.h>
#include <stdlib.h>
#include "charList.h"
#include "charListIO.h"
int main( void )
{
	Node *head = NULL , *n;
	char c;
	while( fscanf( stdin , " %c" , &c )==1 )
	{
		if( !head ) head  = create_node( c );
		else
		{
			n = head;
			while( n->next ) n = n->next;
			add_after( n , c );
		}
	}
	print( head );
	return 0;
}
```

*charList.h*

```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
int length( const Node *head );
```

*charListIO.h*

```
#include "charList.h"
void print( const Node* head );
```

```
>> gcc -std=c99 -Wall -Wextra -g main.c charList.c charListIO.c
In file included from charListIO.h:1:0,
                 from main.c:5:
charList.h:3:16: error: redefinition of struct _Node
 typedef struct _Node
                ^~~~~
...
>>
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "charList.h"
#include "charListIO.h"
int main( void )
{
	Node *head = NULL , *n;
	char c;
	while( fscanf( stdin , " %c" , &c )==1 )
	{
		if( !head ) head  = create_node( c );
		else
		{
			n = head;
			while( n->next ) n = n->next;
			add_after( n , c );
		}
	}
	print( head );
	return 0;
}
```

charList.h

```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
int length( const Node *head );
```

charListIO.h

```
#include "charList.h"
void print( const Node* head );
```

```
>> gcc -std=c99 -Wall -Wextra -g main.c charList.c charListIO.c
In file included from charListIO.h:1:0,
                 from main.c:5:
charList.h:3:16: error: redefinition of struct _Node
 typedef struct _Node
                ^~~~~
...
>>
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "charList.h"
#include "charListIO.h"
int main( void )
{
	Node *head = NULL , *n;
	char c;
	while( fscanf( stdin , " %c" , &c )==1 )
	{
		if( !head ) head  = create_node( c );
		else
		{
			n = head;
			while( n->next ) n = n->next;
			add_after( n , c );
		}
	}
	print( head );
	return 0;
}
```

charList.h

```
#ifndef charList_included
#define charList_included
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
int length( const Node *head );
#endif // charList_included
```

charListIO.h

```
#ifndef charListIO_included
#define charListIO_included
#include "charList.h"
void print( const Node *head );
#endif // charListIO_included
```

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "charList.h"
#include "charListIO.h"
int main( void )
{
	Node *head = NULL , *n;
	char c;
	while( fscanf( stdin , " %c" , &c )==1 )
	{
		if( !head ) head  = create_node( c );
		else
		{
			n = head;
			while( n->next ) n = n->next;
			add_after( n , c );
		}
	}
	print( head );
	return 0;
}
```

charList.h

```c
#ifndef charList_included
#define charList_included
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
int length( const Node *head );
#endif // charList_included
```

charListIO.h

```c
#ifndef charListIO_included
#define charListIO_included

*head );
included
```

```
>> gcc -std=c99 -Wall -Wextra -g main.c charList.c charListIO.c
>> ./a.out
b c d ae
```

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "charList.h"
#include "charListIO.h"
int main( void )
{
	Node *head = NULL , *n;
	char c;
	while( fscanf( stdin , " %c" , &c )==1 )
	{
		if( !head ) head  = create_node( c );
		else
		{
			n = head;
			while( n->next ) n = n->next;
			add_after( n , c );
		}
	}
	print( head );
	return 0;
}
```

charList.h

```c
#ifndef charList_included
#define charList_included
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
int length( const Node *head );
#endif // charList_included
```

charListIO.h

```c
#ifndef charListIO_included
#define charListIO_included

*head );
included
```

```
>> gcc -std=c99 -Wall -Wextra -g main.c charList.c charListIO.c
>> ./a.out
b c d ae
 b c d a e
>>
```

main.c

```c
#include <stdio.h>
#include <stdli
#include "char
#include "char
int main( void )
{
    Node *head = NULL , *n;
    char c;
    while( fscanf( stdin , " %c" , &c )==1 )
    {
        if( !head ) head  = create_node( c );
        else
        {
            n = head;
            while( n->next ) n = n->next;
            add_after( n , c );
        }
    }
    print( head );
    return 0;
}
```

Problems with the code:

- We allocate but don’t deallocate
- The characters are stored in the order they were read, not in alphabetical order

charList.h

```c
#ifndef charList_included
    struct _Node *next;
    char value;
} Node;

Node *create_node( char c );
int add_after( Node *n , char c );
int length( const Node *head );
#endif // charList_included
```

charListIO.h

```c
#ifndef charListIO_included
#define charListIO_included
*head );
included
```

```
>> gcc -std=c99 -Wall -Wextra -g main.c charList.c charListIO.c
>> ./a.out
b c d ae
 b c d a e
>>
```

# Outline

- Exercise 5-3
- Linked lists
- Review questions

## Review questions

1. Describe the linked list structure by a diagram.

## Review questions

2. Compare arrays and linked lists. Write down their pros and cons.

## Review questions

3. What is a linked list's head? How is it different from a node? Explain.

## Review questions

4. How do you calculate length of a linked list?

## Review questions

5. How do you implement `add_after` of a linked list?

## Exercise 6-1

- Website -> Course Materials -> ex6-1