Intermediate Programming

Day 17

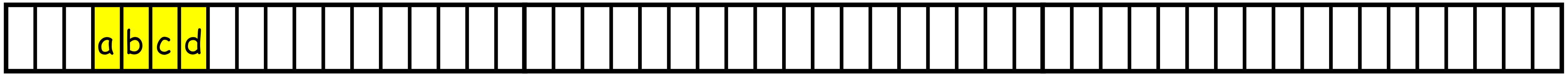
Outline

- 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

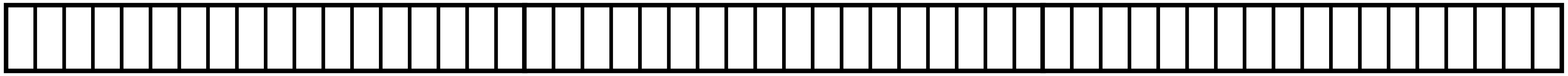
Linked lists

- 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;




…
```

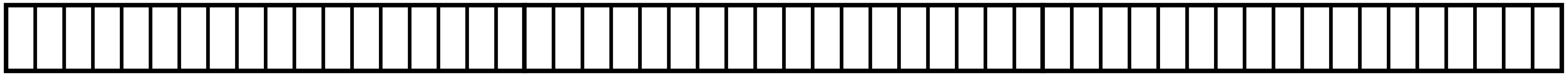

address space

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

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

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



Linked lists

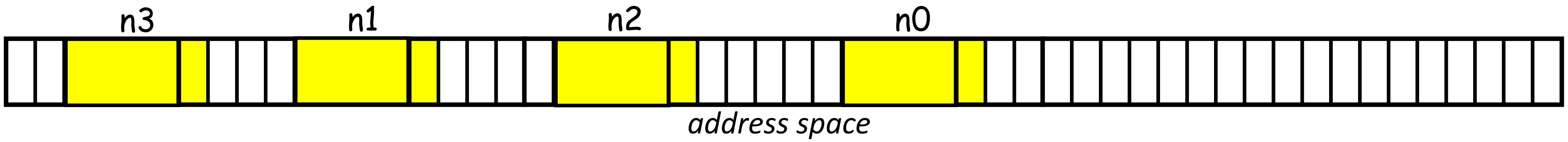
- Arrays:
  - ✓Contiguous memory
    ⇒ Fast (constant time) look-up
  - ✗ Do not support dynamic insertion/deletion
- Linked lists:
  - ✓Support dynamic insertion/deletion
  - ✗ Discontiguous memory
    ⇒ Slow (linear time) look-up
  - ✗ Explicit pointer storage

```
…
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 ) );



…
```



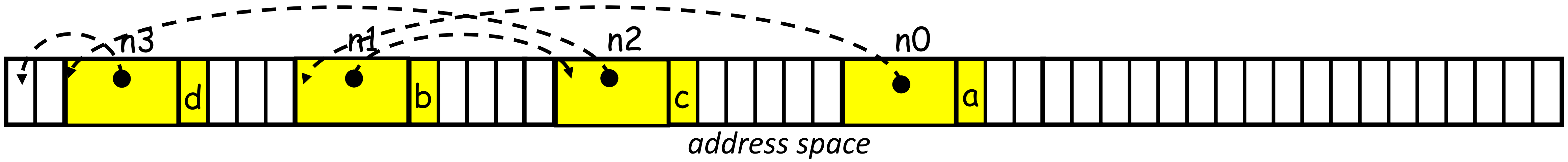
Linked lists

- 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;
…
```



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

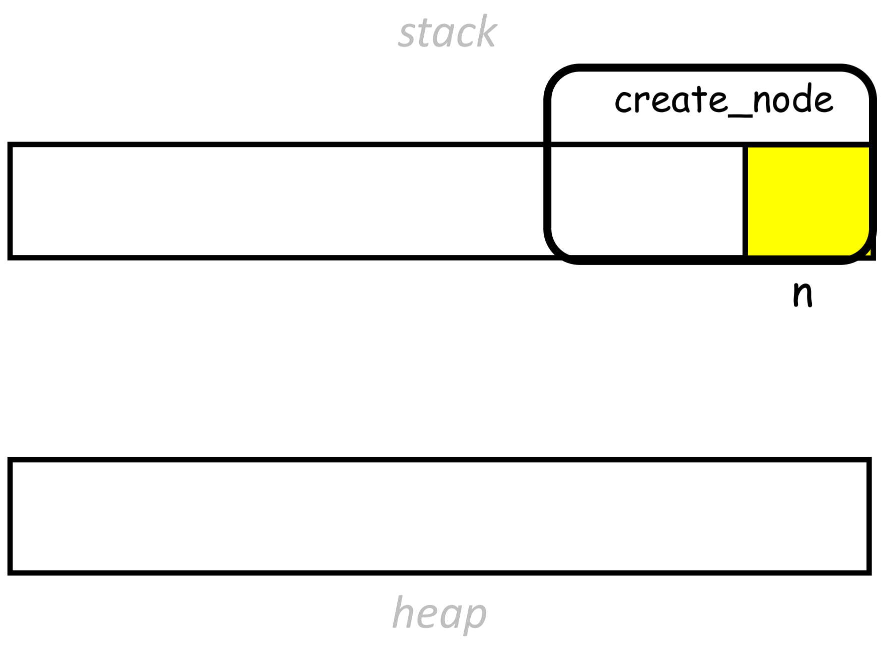
```
typedef struct _Node
{
	struct _Node *next;
	char value;
} Node;

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

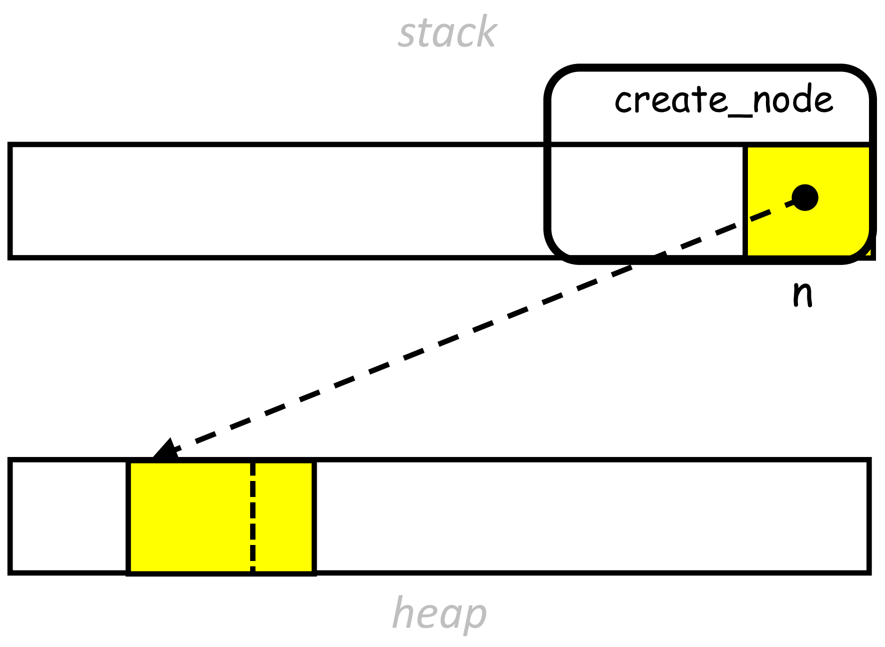
- Create a node
  - Allocate the linked-list element
  - Set its members

```
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

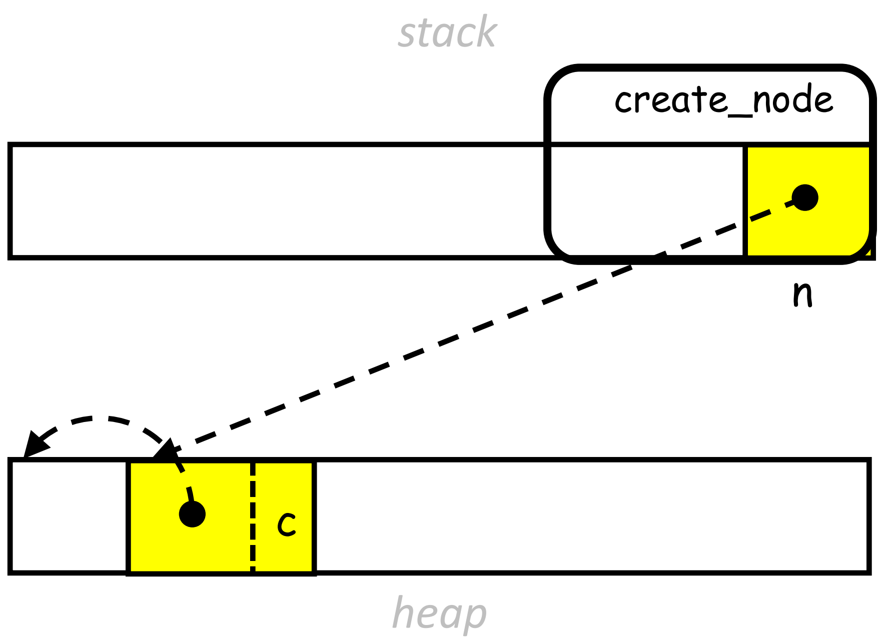
```
typedef struct _Node
{
    struct _Node *next;
    char value;
} Node;

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

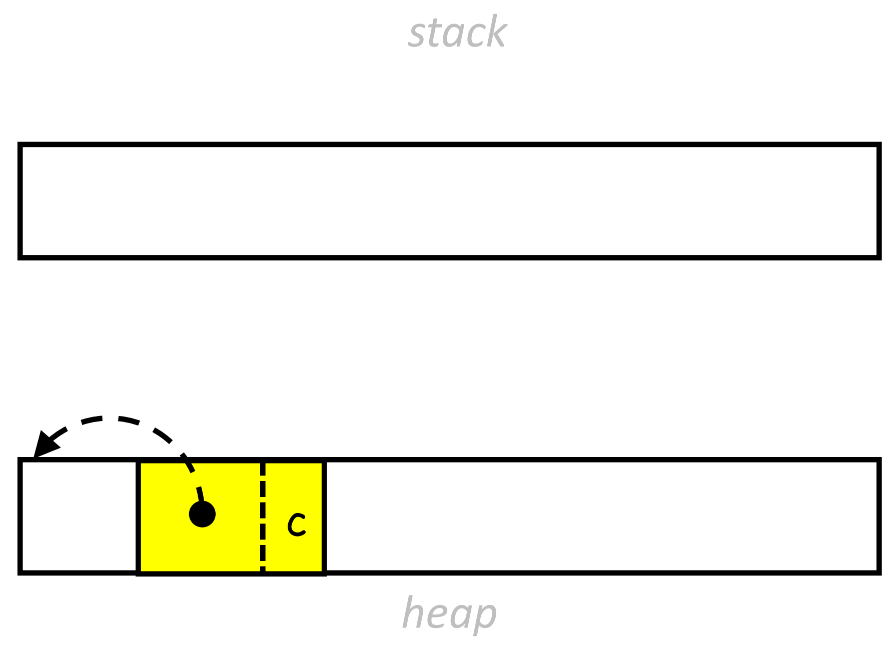
```
typedef struct _Node
{
    struct _Node *next;
    char value;
} Node;

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

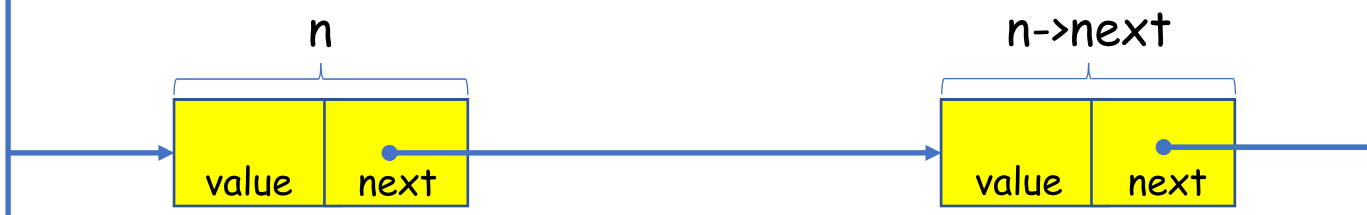
charList.h

```
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

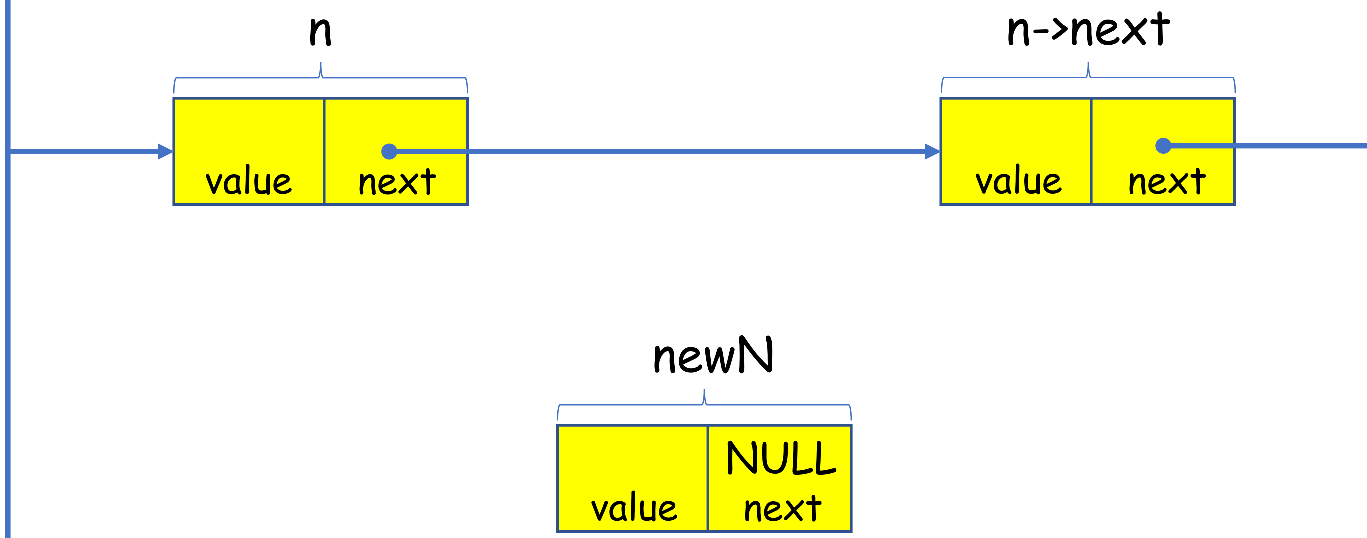
charList.h

```
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**

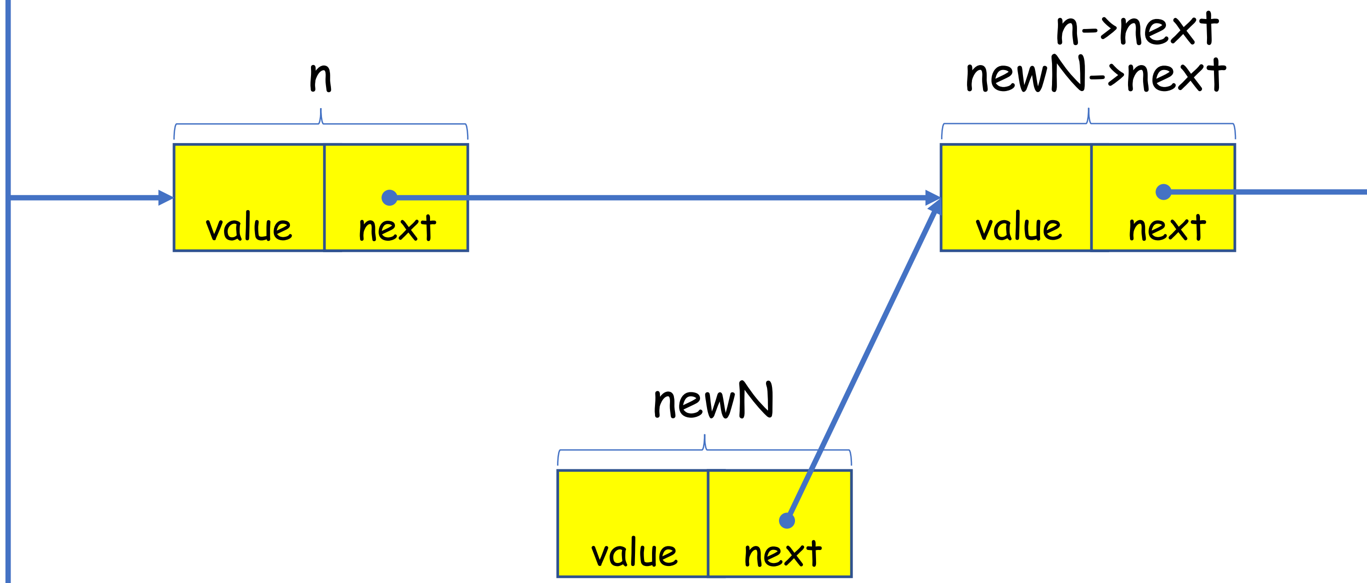
charList.h

```
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**

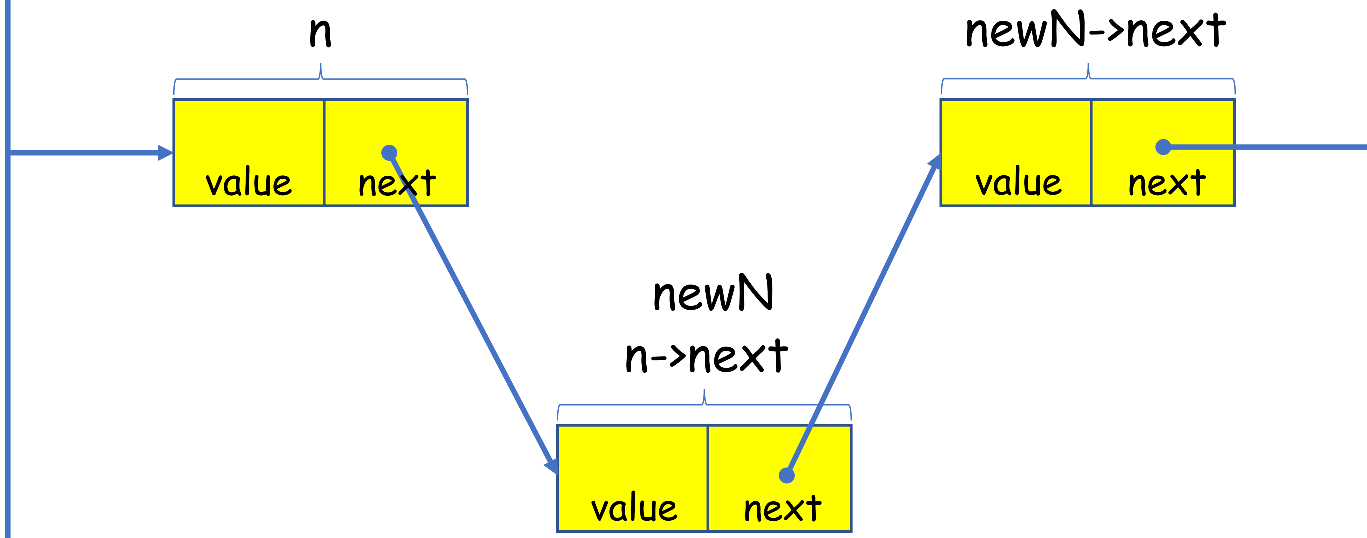
charList.h

```
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

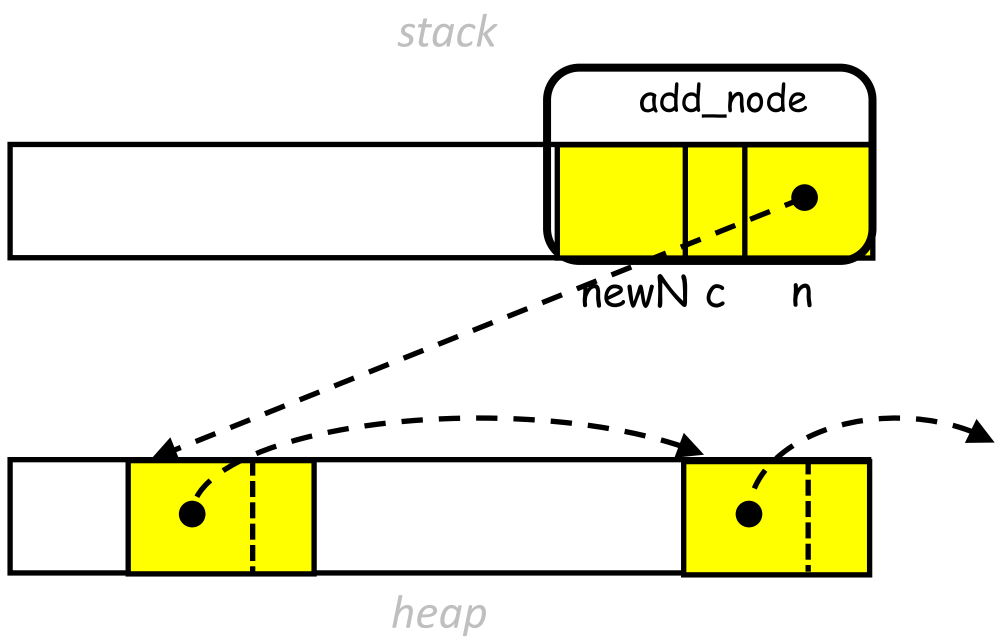
charList.h

```
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

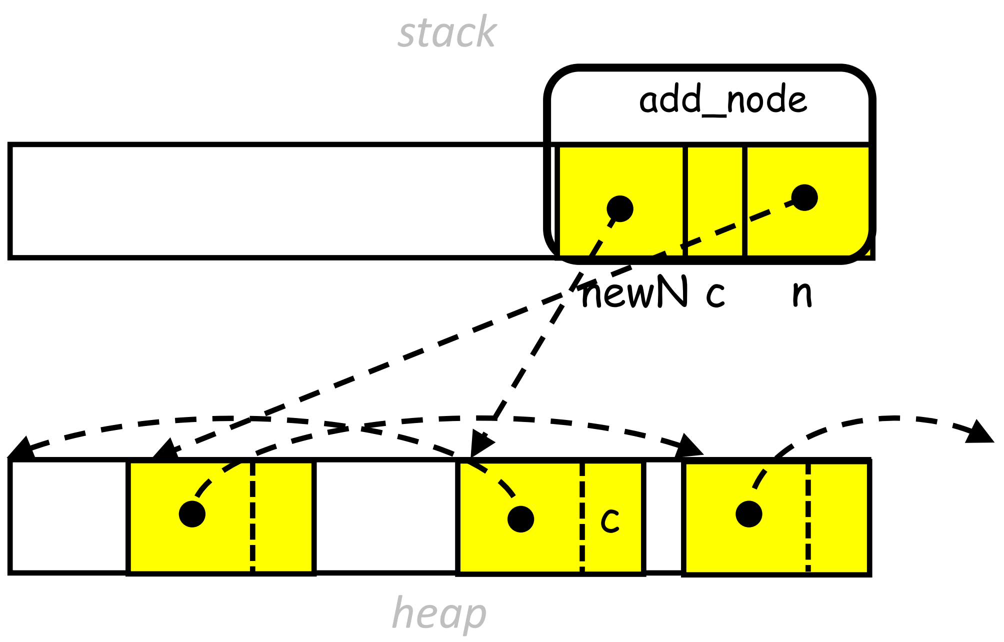
charList.h

```
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

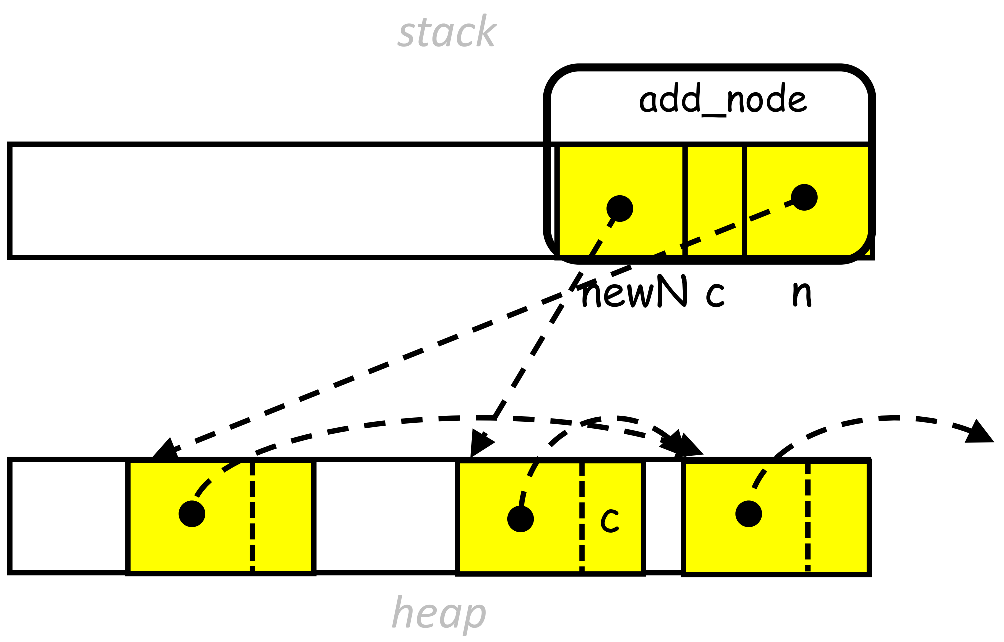
charList.h

```
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

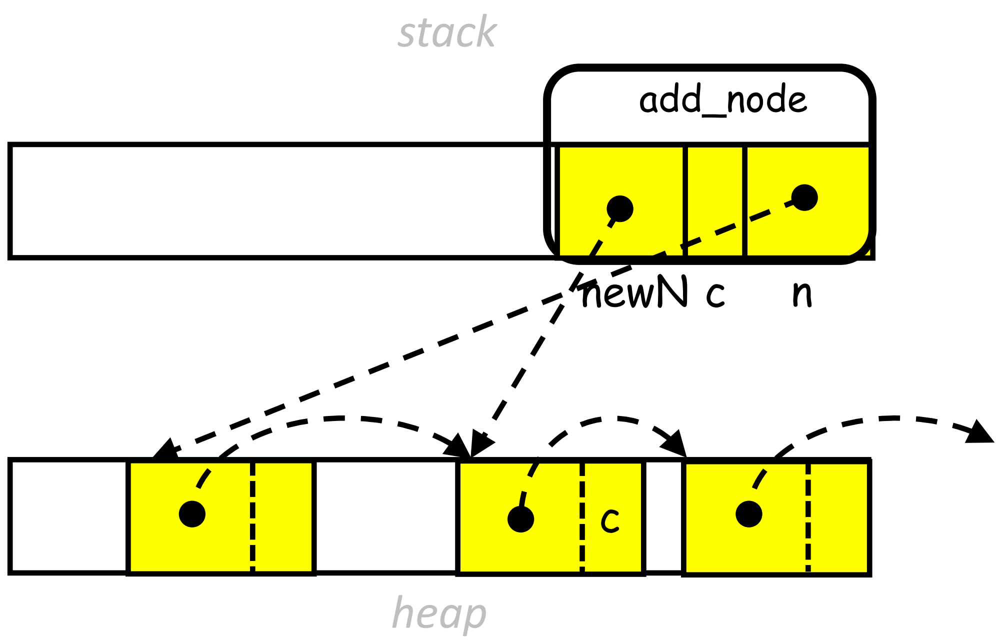
charList.h

```
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

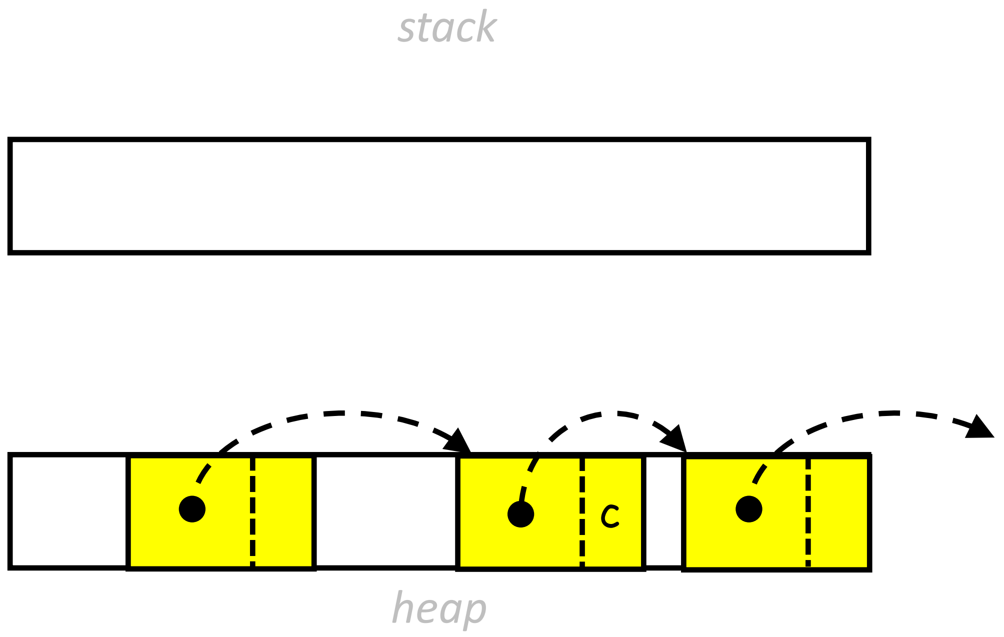
charList.c

```
#include "charList.h"
#include <stdlib.h>
…
int add_after( Node *n , char c )
{
    Elem *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

- 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

```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
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

```c
#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
[illegible]
*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

```
#include <stdio.h>
#include <stdli
#include "charl
#include "charl
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

```
#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

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

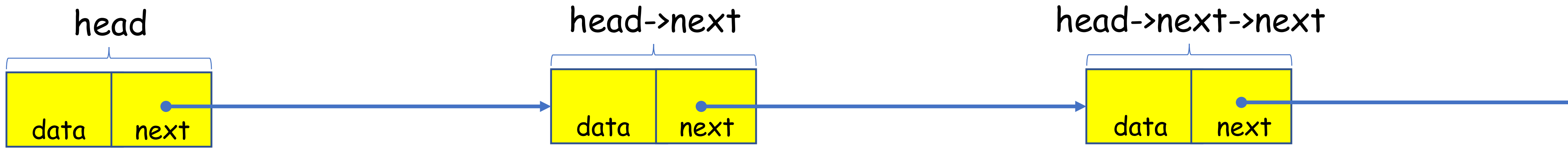
```
>> gcc -std=c99 -Wall -Wextra -g main.c charList.c charListIO.c
>> ./a.out
b c d ae
 b c d a e
>>
```

Outline

- 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.

- 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

Review questions

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

The head is a pointer to the first node in the list.
It could be NULL (if the list is empty) and does not have anything point to it (so the rest of the list is accessible through it).

Review questions

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

```
int length( const Node *head )
{
    int len=0;
    while( head ){ len++ ; head = head->next; }
    return len;
}
```

Review questions

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

```
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;
}
```

Exercise 17

- Website -> Course Materials -> Exercise 17