Structures and linked lists

Part 2
Administration
SO IT BEGINS

THE SECOND GREAT BATTLE OF OUR SEMESTER
Mondays at 1-2pm
LECTURE TOPICS
LECTURE TOPICS

Dynamic linked list allocation
Dynamic array allocation
int size;

/* 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free (array);
int size;

/* 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free (array);
/* 1. declare a pointer to the array */
char *array;

printf("Enter the array length: ");
scanf("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = ‘R’; */

/* 4. free the memory */
free (array);
int size;

/* 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free (array);
/ 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free (array);
*/ 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* e.g., array[i] = 'R';       */

/* 4. free the memory */
free (array);
/* 1. declare a pointer to the array */
char *array;

printf("Enter the array length: ");
scanf("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* e.g., array[i] = 'R';       */

/* 4. free the memory */
free(array);
/ * 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free (array);
/ * 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *)calloc(size, sizeof(char));

/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free (array);
/* 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* e.g., array[i] = 'R';       */

/* 4. free the memory */
free (array);
int size;
/* 1. declare a pointer to the array */
char *array;

printf("Enter the array length: ");
scanf("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free(array);

/* allocates memory block for an array of size elements */
/* e.g., array[i] = 'R'; */

/* free (array); */

memory

number of elements × sizeof(type) in bytes
/ 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free (array);
/* 1. declare a pointer to the array */
char *array;

printf("Enter the array length: ");
scanf("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free(array);
/* 1. declare a pointer to the array */
char *array;

printf("Enter the array length: ");
scanf("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free(array);
/ 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free (array);

returns the address of the first byte in array
return type is void *
/* 1. declare a pointer to the array */
char *array;

printf("Enter the array length: ");
scanf("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R';       */

/* 4. free the memory */
free (array);

returns the address of the first byte in array

return type is void *
must typecast to array type

memory

...
/* 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R';       */

/* 4. free the memory */
free (array);

returns the address of the first byte in array
return type is void *
must typecast to array type
/* 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = ‘R’; */

/* 4. free the memory */
free (array);
/ 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free (array);
int size;

/* 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R'; */

/* 4. free the memory */
free (array);
int size;

/* 1. declare a pointer to the array */
char *array;

printf ("Enter the array length: ");
scanf ("%d", &size);

/* 2. allocate the array in memory */
array = (char *) calloc(size, sizeof(char));

/* 3. use the array normally */
/* e.g., array[i] = 'R';       */

/* 4. free the memory */
free (array);

See Program #9 at c.ihypress.ca/07.php
Dynamic linked list allocation
linked list

an ordered set of data elements, each containing a link to its successor
linked list
an ordered set of data elements, each containing
a link to its successor
and sometimes its predecessor
start
start

structure

1

2

3

×
start

structure
data fields
Start

Structure

Data fields

Address to next element
typedef struct elephant
{
    char name[10];
    int weight;
    struct elephant* next;
}elephant;
typedef struct elephant
{
    char name[10];
    int weight;
    struct elephant* next;
} elephant,
elephant *current, *first;
int response;

/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;

printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf ("%d", &response)

while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
current->next =
    current = current->next;
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);
    printf("\nAdd another? (y=1/n=0) ");
    scanf ("%d", &response);
}
current->next = NULL;
elephant *current, *first;
int response;

/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;

printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response);

while (response) /*while response is 1 (yes) */{
    /* allocate node and change current pointer */
    current->next = (elephant *)calloc(1,sizeof(elephant));
current = current->next;
    printf("Elephant name? ");
    scanf("%s", current->name);
    printf("Elephant weight? ");
    scanf("%d", &current->weight);
    printf("\nAdd another? (y=1/n=0) ");
    scanf("%d", &response);
}
current->next = NULL;
elephant *current, *first;
int response;

/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;

printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)");
scanf ("%d", &response);

while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
current->next =
/* allocate node and change current pointer */
current->next =
}
```c
elephant *current, *first;
int response;

/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;

printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)" );
scanf ("%d", &response)

while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
```
elephant *current, *first;
int response;

/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;

printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)");
scanf ("%d", &response);

while (response) /* while response is 1 (yes) */
{
    /* allocate node and change current pointer */
current->next = (elephant *)calloc(1,sizeof(elephant));
current = current->next;
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);
    printf("\nAdd another? (y=1/n=0)");
    scanf ("%d", &response);
}
elephant *current, *first;
int response;

/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;

printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)");
scanf ("%d", &response);

while (response)  /*while response is 1 (yes) */
{   /* allocate node and change current pointer */
    current->next =
    /* fill node */
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);
    printf("\nAdd another? (y=1/n=0)");
    scanf ("%d", &response);
    current->next = NULL;
}
elephant *current, *first;
int response;

/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;

printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)");
scanf ("%d", &response);

while (response) /*while response is 1 (yes) */ {
    /* allocate node and change current pointer */
current->next = current = current->next;
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);
    printf("\nAdd another? (y=1/n=0)");
    scanf ("%d", &response);
}
current->next = NULL;
elephant *current, *first;
int response;

/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;

printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)" );
scanf ("%d", &response);

while (response) /*while response is 1 (yes) */ {
    /* allocate node and change current pointer */
    current->next =
    current = current->next;
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);
    printf("\nAdd another? (y=1/n=0)" );
    scanf ("%d", &response);
    current->next = NULL;
}
while (response) /*while response is 1 (yes) */ {
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;

    /* fill node */
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);

    printf("\nAdd another? (y=1/n=0) ");
    scanf ("%d", &response);
}

current->next = NULL;
while (response) /*while response is 1 (yes) */ {
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;

    /* fill node */
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);

    printf("\nAdd another? (y=1/n=0)" );
    scanf ("%d", &response);
}

current->next = NULL;
while (response) /* while response is 1 (yes) */ {

    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1, sizeof(elephant));
    current = current->next;

    /* fill node */
    printf("Elephant name? ");
    scanf("%s", current->name);
    printf("Elephant weight? ");
    scanf("%d", &current->weight);

    printf("\nAdd another? (y=1/n=0) ");
    scanf("%d", &response);
}

current->next = NULL;
while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;

    /* fill node */
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);

    printf("\nAdd another? (y=1/n=0)");
    scanf ("%d", &response);
}

current->next = NULL;
while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;

    /* fill node */
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);

    printf("\nAdd another? (y=1/n=0)");
    scanf ("%d", &response);
}

current->next = NULL;
while (response) /*while response is 1 (yes) */ {
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;

    /* fill node */
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);

    printf("\nAdd another? (y=1/n=0) ");
    scanf ("%d", &response);
}

current->next = NULL;
while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;

    /* fill node */
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);

    printf("\nAdd another? (y=1/n=0) ");
    scanf ("%d", &response);
}

current->next = NULL;
while (response) /*while response is 1 (yes) */ {
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
current = current->next;

    /* fill node */
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);

    printf("\nAdd another? (y=1/n=0) ");
    scanf ("%d", &response);
}
current->next = NULL;
while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;

    /* fill node */
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);

    printf("\nAdd another? (y=1/n=0) ");
    scanf ("%d", &response);
}

current->next = NULL;
elephant *current, *first;
int response;

/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;

printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)");
scanf ("%d", &response)

while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
current = current->next;

    /* fill node */
    printf("Elephant name? ");
    scanf ("%s", current->name);
    printf("Elephant weight? ");
    scanf ("%d", &current->weight);

    printf("\nAdd another? (y=1/n=0)");
    scanf ("%d", &response);
}
current->next = NULL;
elephant *current, *first;
int response;
elephant *current, *first;
int response;
elephant *current, *first;
int response;
elephant *current, *first;
int response;
elephant *current, *first;
int response;
/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;
/ * create first node */

```c
first = (elephant*)calloc(1, sizeof(elephant));
current = first;
```
/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;
/* create first node */
first = (elephant*)calloc(1,sizeof(elephant));
current = first;
/* create first node */

first = (elephant*)calloc(1,sizeof(elephant));
current = first;
/ * create first node */

```c
first = (elephant*)calloc(1,sizeof(elephant));
current = first;
```
printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)");
scanf ("%d", &response)
printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)" );
scanf ("%d", &response)
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)");
scanf("%d", &response)
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)" );
scanf("%d", &response)
printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)" );
scanf ("%d", &response)
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response)
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response)
```c
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);
printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response)
```

```
first    current
```

```
"Edna" 450
```
```c
printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response)
```

```

first current

```

```

"Edna" 450

```
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response)

user input: 1

first

Edna 450

current
while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;

    "Edna" 450
}
while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;
}
while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;
}
while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;
}
while (response) /* while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;
}
while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;

    "Edna" 450
while (response) /*while response is 1 (yes)*/ {
    /* allocate node and change current pointer */
    current->next = (elephant *)calloc(1, sizeof(elephant));
    current = current->next;
}

first

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edna</td>
<td>450</td>
</tr>
</tbody>
</table>
```

current
/* fill node */
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response);
/* fill node */
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response);
}
/ * fill node */
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)" );
scanf("%d", &response);
/* fill node */
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response);
}
/* fill node */
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)");
scanf("%d", &response);
}
/ * fill node */

printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf ("%d", &response);

 first

`/* fill node */`

```c
printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)" );
scanf ("%d", &response);
}
```

current
/* fill node */

printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response);

/* fill node */
printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf ("%d", &response);
}
/* fill node */

```c
printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)"));
scanf("%d", &response);
```

User input: 1

```

---

first

```

```

current

```

```

"Edna" 450

```

```

"Elm" 600

---

```
while (response) /*while response is 1 (yes) */ {
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;

    /* Edna */ 450
    current

    /* Elm */ 600
while (response) /*while response is 1 (yes) */ {
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;
}
while (response) /*while response is 1 (yes) */{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;
}
while (response) /* while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;

first

![Diagram of linked list with nodes "Edna" and "Elm"]

current
while (response) /* while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1, sizeof(elephant));
    current = current->next;
}
while (response) /*while response is 1 (yes) */
{
    /* allocate node and change current pointer */
    current->next =
        (elephant *)calloc(1,sizeof(elephant));
    current = current->next;
}
while (response) /* while response is 1 (yes) */ {
    /* allocate node and change current pointer */
    current->next = (elephant *)calloc(1,sizeof(elephant));
    current = current->next;
}
/ * fill node */
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response);
}
/* fill node */
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response);
}
/* fill node */

printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf ("%d", &response);
/* fill node */

printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)");
scanf("%d", &response);

/ * fill node */
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response);
/* fill node */
printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0)");
scanf ("%d", &response);
}
/* fill node */
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response);
/* fill node */
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response);
}
/* fill node */
printf("Elephant name? ");
scanf ("%s", current->name);
printf("Elephant weight? ");
scanf ("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf ("%d", &response);
}
/* fill node */
printf("Elephant name? ");
scanf("%s", current->name);
printf("Elephant weight? ");
scanf("%d", &current->weight);

printf("\nAdd another? (y=1/n=0) ");
scanf("%d", &response);
}
current->next = NULL;
current->next = NULL;
current->next = NULL;
Quiz
typedef struct element
{
    int num;
    struct element* next;
}element;

element *current, *first;
int i;

first = (element*)calloc(1,sizeof(element));
current = first;

for (i = 0; i < 10; i++)
{
    current->next = current->next;
    current = current->next;
}
current->next = NULL;
typedef struct element {
    int num;
    struct element* next;
} element;

element *current, *first;
int i;

first = (element*)calloc(1,sizeof(element));
current = first;

for (i = 0; i < 10; i++)
{
    current->next = (element *)calloc(1,sizeof(element));
    current = current->next;
}
current->next = NULL;
typedef struct element
{
    int num;
    struct element* next;
}element;

element *current, *first;
int i;

first = (element*)calloc(1,sizeof(element));
current = first;

for (i = 0; i < 10; i++)
{
    current->next = (element *)calloc(1,sizeof(element));
    current = current->next;
}

current->next = NULL;
Write a function that returns an integer of 1 if an array of integers is in ascending order and 0 otherwise.
Write a function that returns an integer of 1 if an array of integers is in ascending order and 0 otherwise.

```c
int ascending(int x[], int length)
{
    int i;

    for (i = 1; i < length; i = i + 1)
    {
        if (x[i-1] > x[i])
            return 0;
    }

    return 1;
}
```
Write a function that returns an integer of 1 if an array of integers is in ascending order and 0 otherwise.

```c
int ascending(int x[], int length)
{
    int i;

    for (i = 1; i < length; i = i + 1)
    {
        if (x[i-1] > x[i])
            return 0;
    }

    return 1;
}
```
Write a function that returns an integer of 1 if an array of integers is in ascending order and 0 otherwise.

```c
int ascending(int x[], int length)
{
    int i;

    for (i = 1; i < length; i = i + 1)
    {
        if (x[i-1] > x[i])
            return 0;
    }

    return 1;
}
```
Write a function that returns an integer of 1 if an array of integers is in ascending order and 0 otherwise.

```c
int ascending(int x[], int length)
{
    int i;

    for (i = 1; i < length; i = i + 1)
    {
        if (x[i-1] > x[i])
            return 0;
    }

    return 1;
}
```
Write a function that returns an integer of 1 if an array of integers is in ascending order and 0 otherwise.

```c
int ascending(int x[], int length)
{
    int i;

    for (i = 1; i < length; i = i + 1)
    {
        if (x[i-1] > x[i])
            return 0;
    }

    return 1;
}
```
Write a function that returns an integer of 1 if an array of integers is in ascending order and 0 otherwise.

```c
int ascending(int x[], int length)
{
    int i;

    for (i = 1; i < length; i = i + 1)
    {
        if (x[i-1] > x[i])
            return 1;

    return 1;
}
```
Write a function that returns an integer of 1 if an array of integers is in ascending order and 0 otherwise.

```c
int ascending(int x[], int length)
{
    int i;

    for (i = 1; i < length; i = i + 1)
    {
        if (x[i-1] > x[i])
            return 0;
    }

    return 1;
}
```
Write a function that returns an integer of 1 if an array of integers is in ascending order and 0 otherwise.

```c
int ascending(int x[], int length)
{
    int i;

    for (i = 1; i < length; i = i + 1)
    {
        if (x[i-1] > x[i])
            return 0;
    }

    return 1;
}
```
Write a function that returns an integer of 1 if an array of integers is in ascending order and 0 otherwise.

```c
int ascending(int x[], int length)
{
    int i;

    for (i = 1; i < length; i = i + 1)
    {
        if (x[i-1] > x[i])
            return 0;
    }

    return 1;
}
```
Write a function that takes an integer array input and returns the number of even array elements.
Write a function that takes an integer array input and returns the number of even array elements.

```c
int evenCount(int a[], int length)
{ 
    int count = 0, i;

    for(i = 0; i < length; i++)
    {
        if(a[i] % 2 == 0)
            count++;
    }

    return count;
}
```
Write a function that takes an integer array input and returns the number of even array elements.

```c
int evenCount(int a[], int length)
{
    int count = 0, i;

    for(i = 0; i < length; i++)
    {
        if(a[i] % 2 == 0)
            count++;
    }
    return count;
}
```
Write a function that takes an integer array input and returns the number of even array elements.

```c
int evenCount(int a[], int length)
{
    int count = 0, i;

    for(i = 0; i < length; i++)
    {
        if(a[i] % 2 == 0)
            count++;
    }

    return count;
}
```
Write a function that takes an integer array input and returns the number of even array elements.

```c
int evenCount(int a[], int length)
{
    int count = 0, i;

    for(i = 0; i < length; i++)
    {
        if(a[i] % 2 == 0)
            count++;
    }

    return count;
}
```
Write a function that takes an integer array input and returns the number of even array elements.

```c
int evenCount(int a[], int length) {
    int count = 0, i;

    for(i = 0; i < length; i++)
    {
        if(a[i] % 2 == 0)
            count++;
    }

    return count;
}
```
Write a function that takes an integer array input and returns the number of even array elements.

```c
int evenCount(int a[], int length)
{
    int count = 0, i;

    for(i = 0; i < length; i++)
    {
        if(a[i] % 2 == 0)
            count++;
    }

    return count;
}
```
Write a function that takes an integer array input and returns the number of even array elements.

```c
int evenCount(int a[], int length)
{
    int count = 0, i;

    for(i = 0; i < length; i++)
    {
        if(a[i] % 2 == 0)
            count++;
    }

    return count;
}
```
Write a code fragment that computes the sum of the elements along the diagonal of a square 2D matrix of integers.
Write a code fragment that computes the sum of the elements along the diagonal of a square 2D matrix of integers.
Write a code fragment that computes the sum of the elements along the diagonal of a square 2D matrix of integers.
Write a code fragment that computes the sum of the elements along the diagonal of a square 2D matrix of integers.

```c
int a[3][3] =
    { {1, 4, 3}, {-2, 5, 0}, {8, -33, -8} };
int sum = 0, i;

for (i = 0; i < 3; i++)
{
    sum += array[i][i];
}

printf("The sum along the diagonal is %d\n", sum);
```
Write a code fragment that computes the sum of the elements along the diagonal of a square 2D matrix of integers.

```c
int a[3][3] =
{   {1, 4, 3},
    {-2, 5, 0},
    {8, -33, -8}  
};
int sum = 0, i;

for (i = 0; i < 3; i++)
{
    sum += array[i][i];
}

printf("The sum along the diagonal is %d\n", sum);
```
Write a code fragment that computes the sum of the elements along the diagonal of a square 2D matrix of integers.

```c
int a[3][3] =
    { {1, 4, 3}, {-2, 5, 0}, {8, -33, -8} };
int sum = 0, i;

for (i = 0; i < 3; i++)
{
    sum += array[i][i];
}

printf("The sum along the diagonal is %d\n", sum);
```
Write a code fragment that computes the sum of the elements along the diagonal of a square 2D matrix of integers.

```c
int a[3][3] =
    { {1, 4, 3}, {-2, 5, 0}, {8, -33, -8} };
int sum = 0, i;

for (i = 0; i < 3 i++)
{
    sum += array[i][i];
}

printf("The sum along the diagonal is %d\n", sum);
```
Write a code fragment that computes the sum of the elements along the diagonal of a square 2D matrix of integers.

```c
int a[3][3] = 
    { {1, 4, 3}, {-2, 5, 0}, {8, -33, -8} };  
int sum = 0, i;

for (i = 0; i < 3; i++)
{
    sum += array[i][i];
}

printf("The sum along the diagonal is %d\n", sum);
```
Write a code fragment that computes the sum of the elements along the diagonal of a square 2D matrix of integers.

```c
int a[3][3] =
    { {1, 4, 3}, {-2, 5, 0}, {8, -33, -8} };
int sum = 0, i;

for (i = 0; i < 3; i++)
{
    sum += array[i][i];
}

printf("The sum along the diagonal is %d\n", sum);
```
Write a code fragment that computes the sum of the elements along the diagonal of a square 2D matrix of integers.

```c
int a[3][3] =
    { {1, 4, 3}, {-2, 5, 0}, {8, -33, -8} };
int sum = 0, i;

for (i = 0; i < 3; i++)
{
    sum += array[i][i];
}

printf("The sum along the diagonal is %d\n", sum);
```
Write a code fragment that determines whether a character array contains a palindrome.
palindrome

a word, sentence, verse or number that reads the same backward or forward
palindrome
a word, sentence, verse or number that reads the same backward or forward

desserts I stressed
palindrome
a word, sentence, verse or number that reads the same backward or forward
desserts I stressed
1221
Write a code fragment that determines whether a character array contains a palindrome.
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' };  
int flag = 1;

for (int i = 0; i < LENGTH/2; i++) {
    if (array[i] != array[LENGTH - 1 - i]) {
        flag = 0;
    }
}

if (flag)
    printf("Array is a palindrome.\n");
else
    printf("Array is not a palindrome.");
```
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' }; int flag = 1;

for (int i = 0; i < LENGTH/2; i++){
    if (array[i] != array[LENGTH - 1 - i]){ // Corrected the index calculation
        flag = 0;
    }
}

if (flag)
    printf("Array is a palindrome.\n");
else
    printf("Array is not a palindrome.\n");
```
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' };  
int flag = 1;

for (int i = 0; i < LENGTH/2; i++){
    if (array[i] != array[LENGTH - 1 - i]){  
        flag = 0;
    }
}

if (flag)
    printf("Array is a palindrome.\n");
else
    printf("Array is not a palindrome.");
```
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' }; int flag = 1;

for (int i = 0; i < LENGTH/2; i++){
    if (array[i] != array[LENGTH - 1 - i]){
        flag = 0;
    }
}

if (flag)
    printf("Array is a palindrome.\n");
else
    printf("Array is not a palindrome.");
```
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' };  
int flag = 1;  

for (int i = 0; i < LENGTH/2; i++){
    if (array[i] != array[LENGTH - 1 - i]){  
        flag = 0;  
    }
}

if (flag)
    printf("Array is a palindrome.\n");
else
    printf("Array is not a palindrome.");
```
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' }; int flag = 1;
for (int i = 0; i < LENGTH/2; i++){
    if (array[i] != array[LENGTH - 1 - i]){flag = 0;}
}
if (flag)
    printf("Array is a palindrome.\n");
else
    printf("Array is not a palindrome.");
```
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' };  

for (int i = 0; i < LENGTH/2; i++) {  
    if (array[i] != array[LENGTH - 1 - i]) {  
        flag = 0;  
    }  
}  

if (flag)  
    printf("Array is a palindrome.\n");  
else  
    printf("Array is not a palindrome.");
```
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' }; int flag = 1;

for (int i = 0; i < LENGTH/2; i++){
    if (array[i] != array[LENGTH - 1 - i]){
        flag = 0;
    }
}

if (flag)
    printf("Array is a palindrome.\n");
else
    printf("Array is not a palindrome.");
```
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' }; int flag = 1;

for (int i = 0; i < LENGTH/2; i++){
    if (array[i] != array[LENGTH - 1 - i]){flag = 0;
}

if (flag)
    printf("Array is a palindrome.\n"); else
    printf("Array is not a palindrome.");
```
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' }; int flag = 1;

for (int i = 0; i < LENGTH/2; i++){
    if (array[i] != array[LENGTH - 1 - i]){ flag = 0; }
}

if (flag) printf("Array is a palindrome.\n"); else printf("Array is not a palindrome.");
```
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' }; int flag = 1;

for (int i = 0; i < LENGTH/2; i++){
    if (array[i] != array[LENGTH - 1 - i]){flag = 0;}
}

if (flag)
    printf("Array is a palindrome.\n");
else
    printf("Array is not a palindrome.");
```
Write a code fragment that determines whether a character array contains a palindrome.

```c
char array[LENGTH] = { 'A', 'B', 'B', 'A' }; int flag = 1;

for (int i = 0; i < LENGTH/2; i++){
    if (array[i] != array[LENGTH - 1 - i]){  
        flag = 0;
    }
}

if (flag)
    printf("Array is a palindrome.\n");
else
    printf("Array is not a palindrome.");
```
/ * computes the conversion between Celsius (C) and Fahrenheit (F), where F = (C*9/5)+32 */
void Cel2Fah(double celc, double *fahr) {
    *fahr = celc*9/5 + 32;
}

int main (void) {
    double c, f;

    printf("Enter the temperature in Celsius: ");
    scanf("%lf", &c);

    Cel2Fah(c, &f);

    printf("The temperature is %lfF\n", f);
}
/* computes the conversion between Celsius (C) and Fahrenheit (F), where F = (C*9/5)+32 */

void Cel2Fah(double celc, double *fahr) {
    *fahr = celc*9/5 + 32;
}

int main (void) {
    double c, f;

    printf("Enter the temperature in Celsius: ");
    scanf("%lf", &c);
    Cel2Fah(c, &f);

    printf("The temperature is %lfF\n", f);
}

/* computes the conversion between Celsius (C) and Fahrenheit (F), where F = (C*9/5)+32 */

void Cel2Fah(double celc, double *fahr) {
    *fahr = celc*9/5 + 32;
}

int main (void) {
    double c, f;

    printf("Enter the temperature in Celsius: ");
    scanf("%lf", &c);
    Cel2Fah(c, &f);
    printf("The temperature is %lfF\n", f);
}
/* computes the conversion between Celsius (C) and Fahrenheit (F), where F = (C*9/5)+32 */

void Cel2Fah(double celc, double *fahr)
{
        *fahr =  celc*9/5 + 32;
}

int main (void)
{
        double c, f;

        printf("Enter the temperature in Celsius: ");
        scanf("%lf", &c);

        Cel2Fah(c, &f);

        printf("The temperature is %lfF\n", f);
}