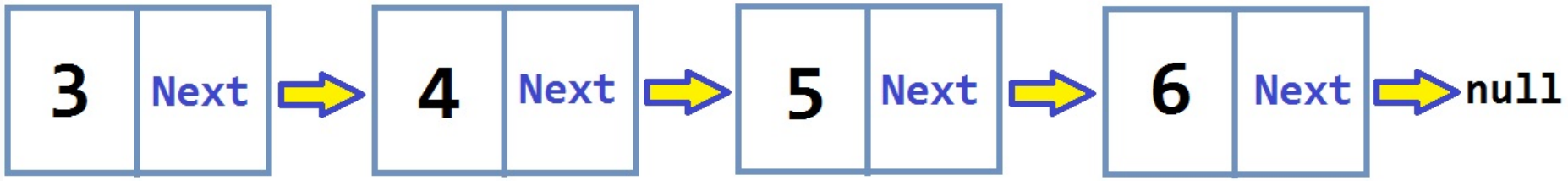
Programming Paradigms

CT331 Week 4 Lecture 3

Finlay Smith

finlay.smith@nuigalway.ie

Linked Lists


3
Next
4
Next
5
Next
6
Next
null

Linked Lists

Pros:

- Dynamic structure. Grown and pruned as data changes.
- Insertion + deletion is easy.
- Can implement other structures easily.
- No defined size.
- Add/remove from middle of list easily.

Cons:

- More memory used than an array for fixed number of elements.
- Sequentially accessed, can’t jump to nth element easily.
- Nodes not stored contiguously, increases memory read time.
- Reverse traverse is cumbersome.

LL Operations

1. Create
2. Traverse
3. Insert (after / before*)
4. Delete (after / current*)
5. Get first element
6. Get rest of the list

*Possible but with potential issues.

Linked list code – available with assignment

```
typedef struct listElementStruct{
  char* data;
  size_t size;
  struct listElementStruct* next;
} listElement;
```

Linked list code

```
//Creates a new linked list element with given content of size
//Returns a pointer to the element
listElement* createEl(char* data, size_t size){
  listElement* e = malloc(sizeof(listElement));
  if(e == NULL){
    //malloc has had an error
    return NULL; //return NULL to indicate an error.
  }
```

Linked list code

```
char* dataPointer = malloc(sizeof(char)*size);
if(dataPointer == NULL){
  //malloc has had an error
  free(e); //release the previously allocated memory
  return NULL; //return NULL to indicate an error.
}
```

Linked list code

```
  strcpy(dataPointer, data);
  e->data = dataPointer;
  e->size = size;
  e->next = NULL;
  return e;
}
```

Linked list code

```
void traverse(listElement* start){
  listElement* current = start;
  while(current != NULL){
    printf("%s\n", current->data);
    current = current->next;
  }
}
```

Linked list code

```
listElement* insertAfter(listElement* el, char* data, size_t size){
  listElement* newEl = createEl(data, size);
  listElement* next = el->next;
  newEl->next = next;
  el->next = newEl;
  return newEl;
}
```

Linked list code

```
void deleteAfter(listElement* after){
  listElement* delete = after->next;
  listElement* newNext = delete->next;
  after->next = newNext;
  //need to free the memory because we used malloc
  free(delete->data);
  free(delete);
}
```