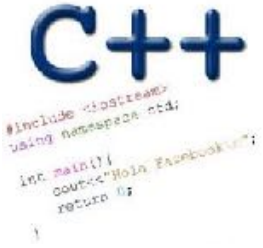
REVIEW POINTERS, DYNAMIC MEMORY LINKED LISTS

Problem Solving with Computers-II



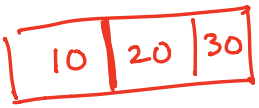
Have you implemented a linked-list before?

A. Yes
B. No

Suppose we were storing a sequence of numbers 10, 20, 30

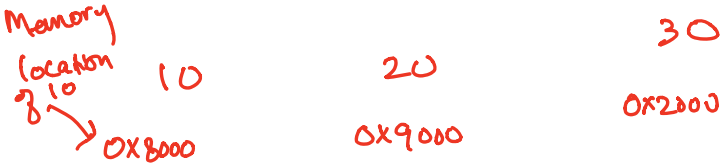
Option 1: Store the numbers in an array

```
int arr[] = {10, 20, 30};
```

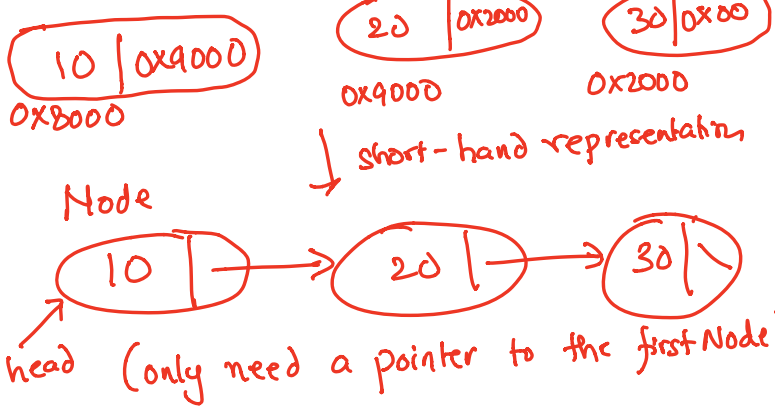


In this case the numbers are stored "next to each other" in memory.

Option 2: Store the numbers at different locations in memory



Since the data is no longer in contiguous memory locations, we need a way to explicitly store not only the data, but also the location of the next number in the sequence (This is the key idea behind a linked list)



A node in a linked-list comprises of the data & a pointer to the next node

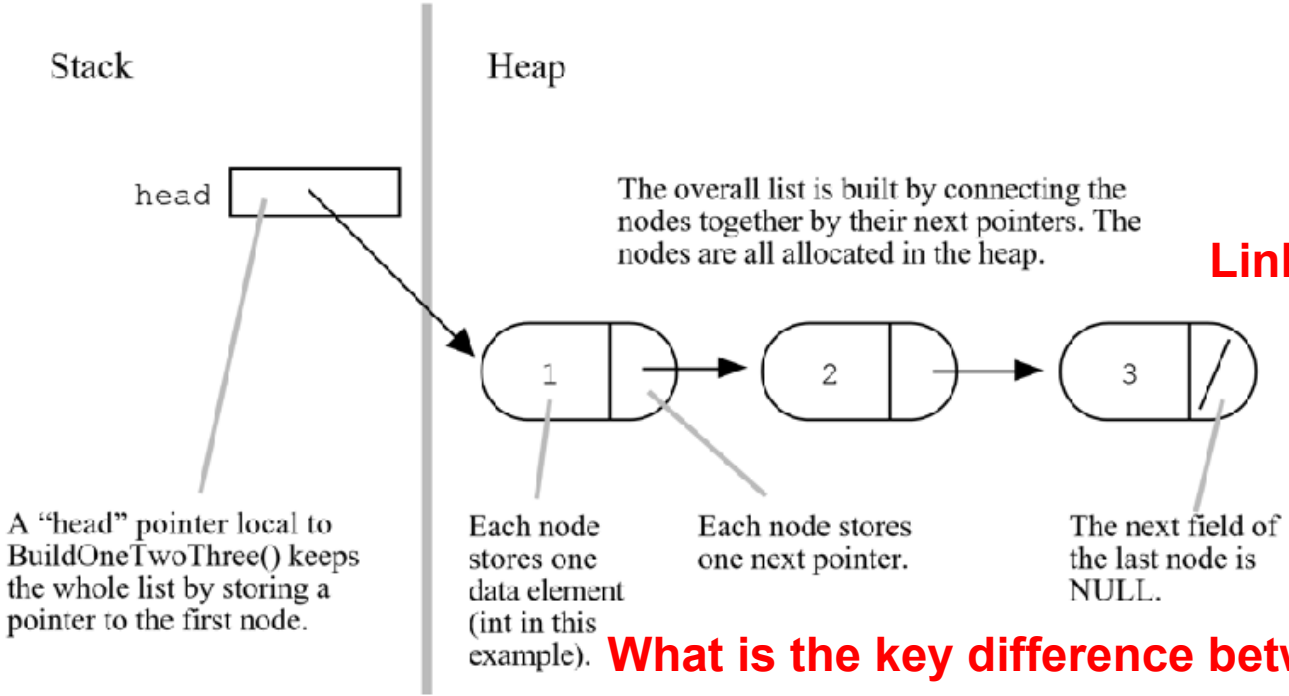
```
struct Node {
    int data;
    Node * next;
};
// You may also represent a node as a class
```

Representing a node in code

Linked Lists

Array List

The Drawing Of List {1, 2, 3}



Linked List

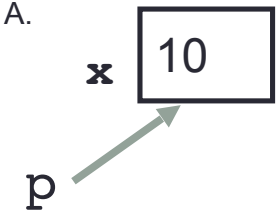
What is the key difference between these?

Review: pointers

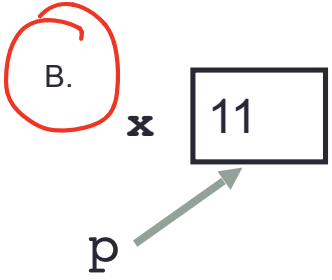
```
int *p, x = 10;
p = &x;
*p = *p + 1;
```

Q: Which of the following pointer diagrams best represents the outcome of the above code?

A.



B.



C. Neither, the code is incorrect

Pointers

- Pointer: A variable that contains the address of another variable
- Declaration: *type* * pointer_name;

```
int *p; // p stores the address of an int
```

What is outcome of the following code?

```
cout<<*p;
```

A. Random number
B. Undefined behavior
C. Null value

Dereferencing a null pointer or a pointer with junk value is likely to result in a segfault.

How do we initialize a pointer?

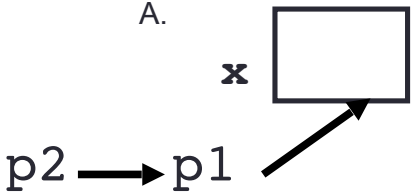
```
int *p = nullptr;
```

Review: Pointer assignment

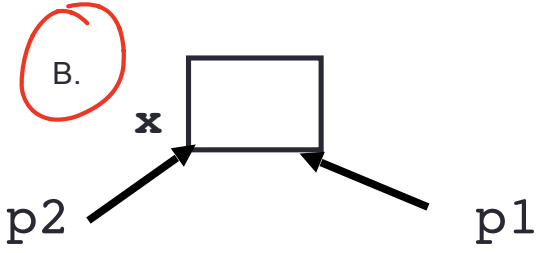
```
int *p1, *p2, x;
p1 = &x;
p2 = p1;
```

Q: Which of the following pointer diagrams best represents the outcome of the above code?

A.



B.



C. Neither, the code is incorrect

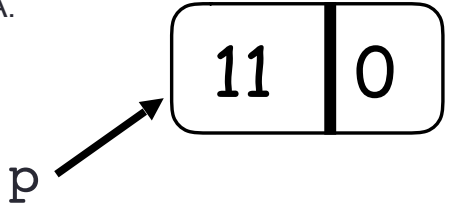
Review: Pointers to structs

```
Node x = {10, nullptr};
Node *p = &x;
p->data = p->data +1;
p = p->next; // p = p→next;
```

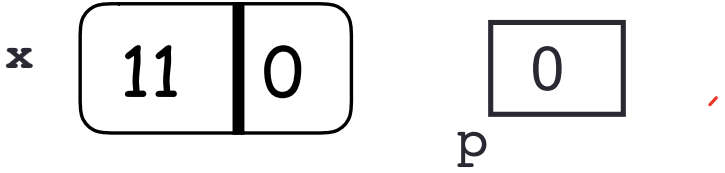
```
struct Node {
    int data;
    Node *next;
};
```

Q: Which of the following pointer diagrams best represents the outcome of the above code?

A.



B.



C. Neither, the code is incorrect

Dynamic memory allocation

- To allocate memory on the heap use the 'new' operator
- To free the memory use delete

```
int *p= new int;
delete p;
```

Avoid code like this:

```
int* createInt(){
   int x = 10;
   return &x;
   }
```

x is a local variable on the stack

It is removed from memory after the function returns

```
int* createIntOnHeap(){
    int *p= new int;
    return p;
}
```

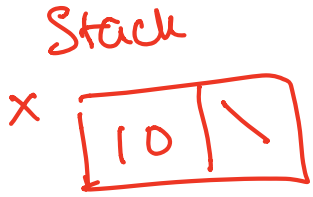
Dynamic memory allocation

- To allocate memory on the heap use the 'new' operator
- To free the memory use delete

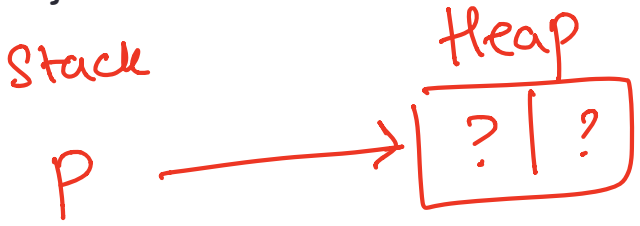
```
int *p= new int;
delete p;
```

```
Node* createNode(){
   Node x = {10, nullptr};
   return &x;
   }
```

x is removed from the stack after the function returns




```
Node* createNodeOnHeap(){
   Node *p = new Node;
   return p;
}
```



Create a two node list

- Define an empty list
- Add a node to the list with data = 10, then 20;

```
struct Node {
    int data;
    Node *next;
};
```

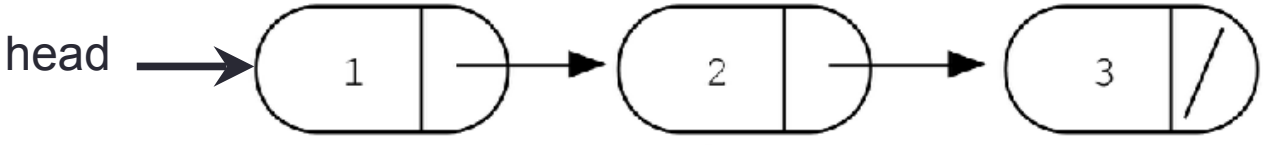
```
Node * head = null ptr; //empty list

head = new Node;
head->data = 10
  head->next = new Node;
  head->next->data = 20;
  head->next->next = 10;
```

Although we can create a two node list like this, you should really have a function to insert new Nodes.

Accessing elements of a list

```
struct Node {
      int data;
      Node *next;
};
```

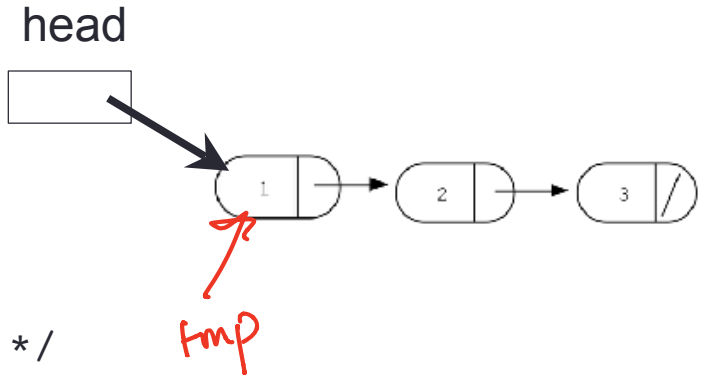


Assume the linked list has already been created, what do the following expressions evaluate to?

1. head->data
2. head->next->data
3. head->next->next->data
4. head->next->next->next->data

A. 1
B. 2
C. 3
D. NULL
E. Run time error

Iterating through the list




```
void printElements(Node* head) {
   /* Print the values in the list */
}
```

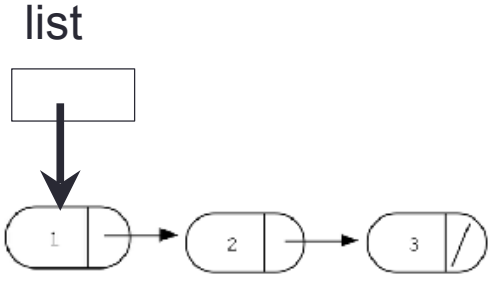
Node * tmp = head;
while tmp {
cout << tmp → data << " " ; //Process this node
tmp = tmp → next; // Make tmp point to the next node in the list
}

Clear the list

```
Node* clearList(Node* head) {
   /* Free all the memory that was created on the heap*/




}
```



Questions you must ask about any data structure:

- **What operations does the data structure support?**
  A linked list supports the following operations:
  1. Insert (a value)
  2. Delete (a value)
  3. Search (for a value)
  4. Min
  5. Max
  6. Print all values
- **How do you implement the data structure?**
- **How fast is each operation?**

Linked-list as an Abstract Data Type (ADT)

```
class IntList {
public:
    IntList();                  // constructor
    ~IntList();                 // destructor
    // other methods
private:
    // definition of Node structure
    struct Node {
        int info;
        Node *next;
    };
    Node *head; // pointer to first node
};
```

Next time

- More linked list with classes