REVIEW POINTERS, DYNAMIC MEMORY LINKED LISTS

Problem Solving with Computers-II

Have you implemented a linked-list before?

A. Yes
B. No

Linked Lists

Array List

The Drawing Of List {1, 2, 3}

Stack

Heap

head

The overall list is built by connecting the nodes together by their next pointers. The nodes are all allocated in the heap.

Linked List

1

2

3

A "head" pointer local to BuildOneTwoThree() keeps the whole list by storing a pointer to the first node.

Each node stores one data element (int in this example).

Each node stores one next pointer.

The next field of the last node is NULL.

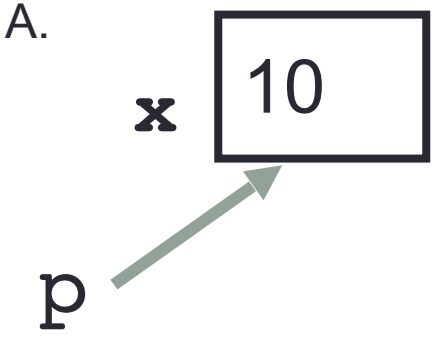
What is the key difference between these?

Review: pointers

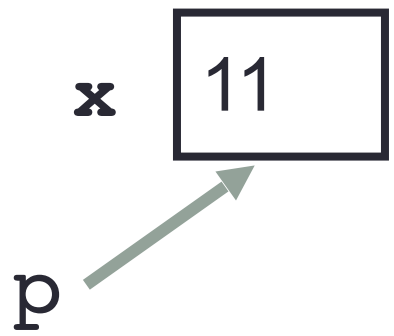
```
int *p, x = 10;
p = &x;
*p = *p + 1;
```

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

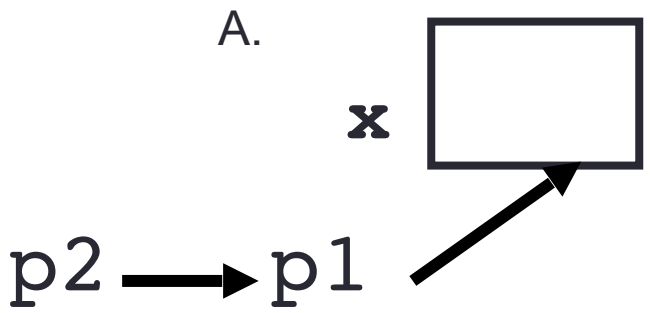
How do we initialize a pointer?

Review: Pointer assignment

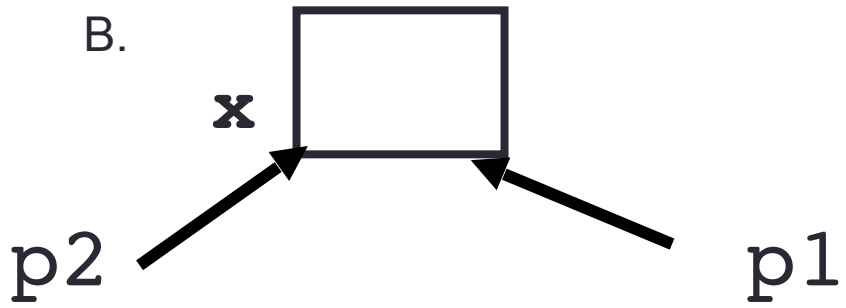
```
int *p1, *p2, x;
p1 = &x;
p2 = p1;
```

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

A.



B.



C. Neither, the code is incorrect

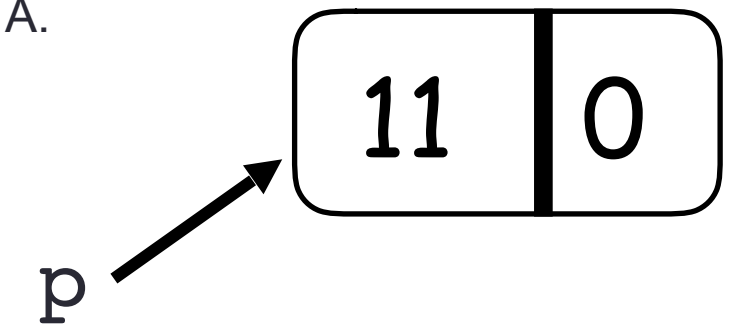
Review: Pointers to structs

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

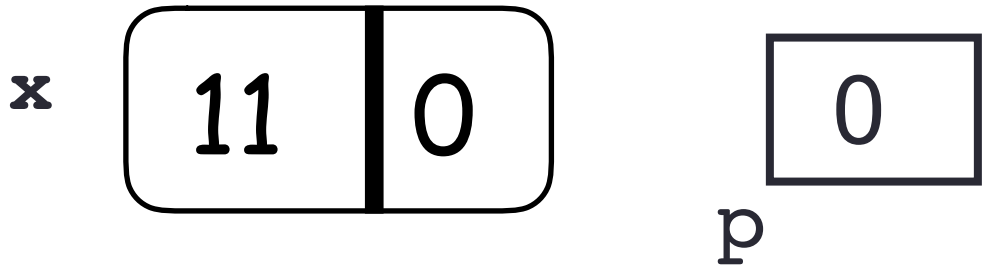
```
struct Node {
    int data;
    Node *next;
};
```

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;
```

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

```
int* createIntOnHeap(){




   }
```

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;
   }
```

```
Node* createNodeOnHeap(){




   }
```

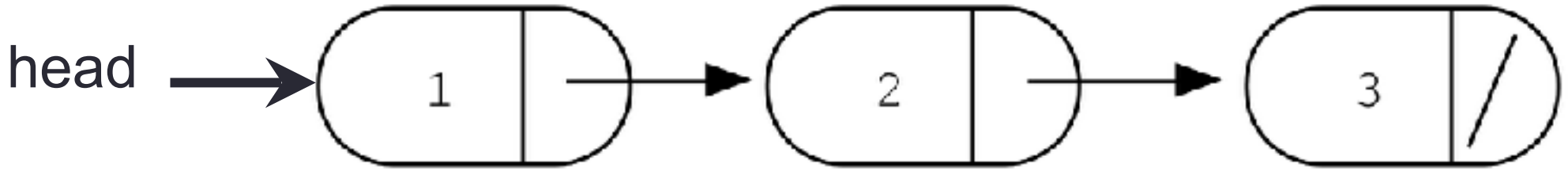
Create a two node list

- Define an empty list
- Add a node to the list with data = 10

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

Accessing elements of a list

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

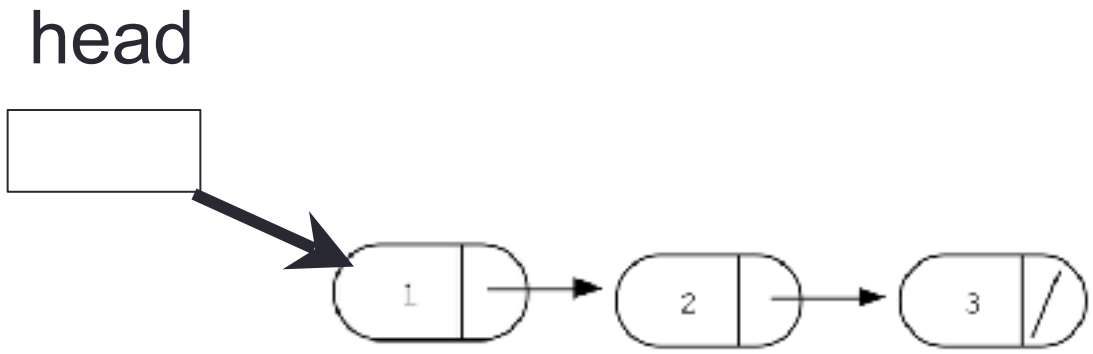


Assume the linked list has already been created, what do the following expressions evaluate to?

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

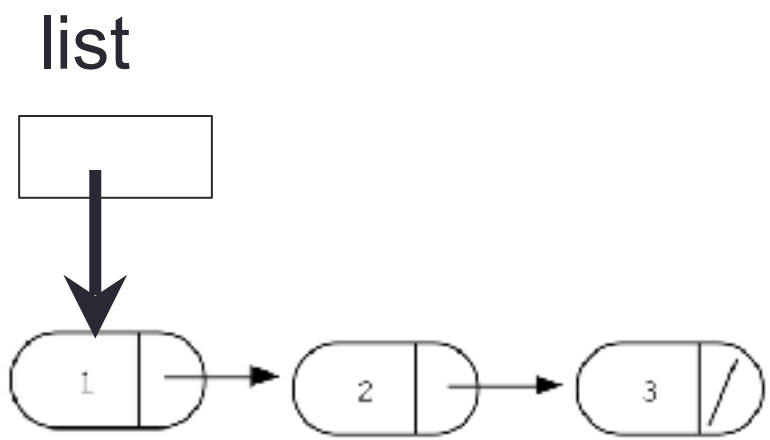
Iterating through the list




```
int lengthOfList(Node * head) {
    /* Find the number of elements in the list */




}
```

Deleting the list

```
Node* freeLinkedList(Node * list) {
    /* 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

- **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