ECE 220

Lecture x001A
Wrap up interrupts & examples

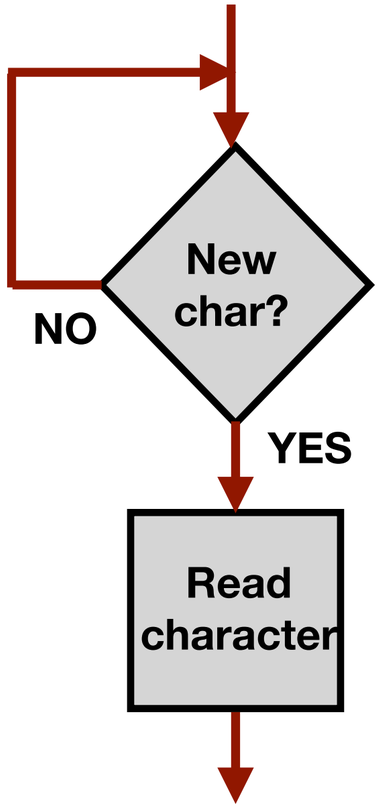
Recap + reminders

- Final exam on 05/13
  - Conflict exam requests due 05/06
- Last day of office hours - 05/06
- Programming competition
- FLEX forms now available

- Extra credit quiz available
  - Extra credit based on score (35 points)
  - Quiz total still capped
- Extra credit survey

Recap - Interrupts

- Polling based I/O vs. Interrupt driven I/O
  - Need to save state of program to be able to resume later
- Interrupts similar to TRAP commands
  - INTerrupt Vector ~ TRAP vector
  - RTI used to return



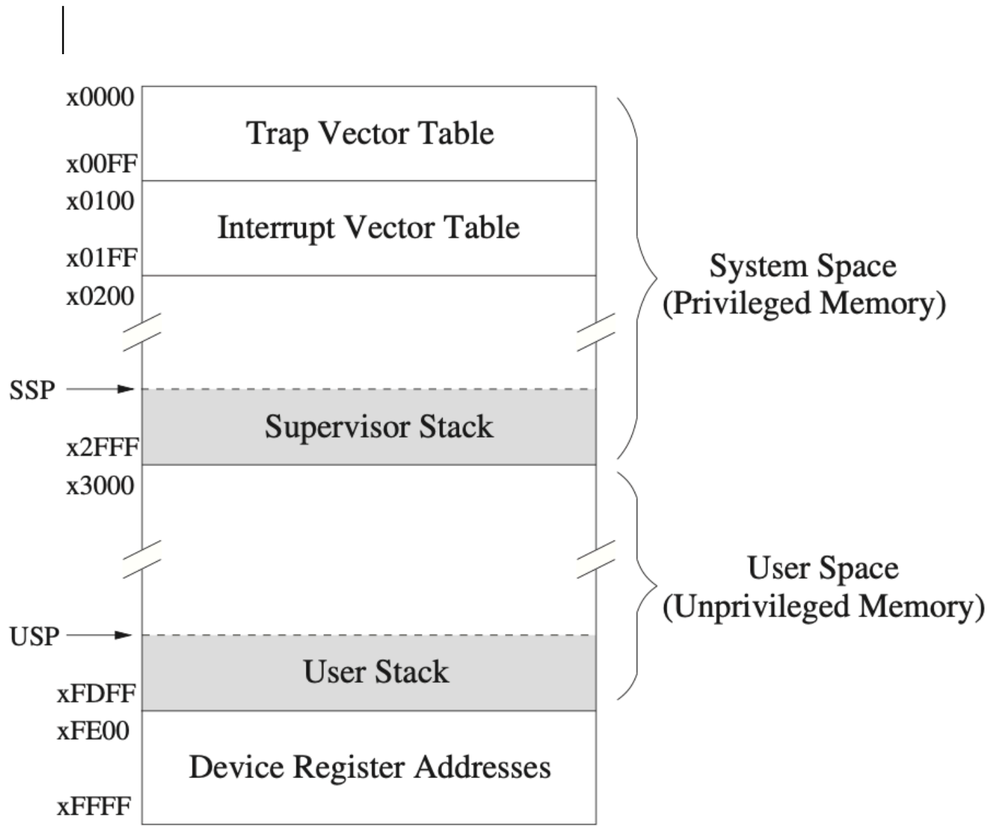
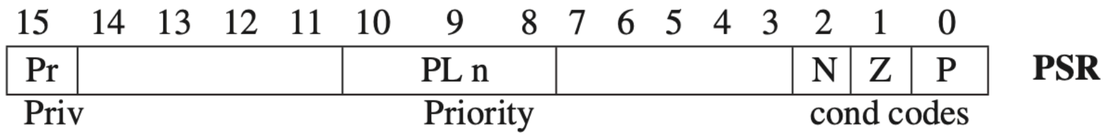


Figure A.1 - P&P 3rd Ed.



Recap - Interrupts

- What needs to be saved?
  - PC, PSR
  - R6 ⇔ Saved_USP, Saved_SSP

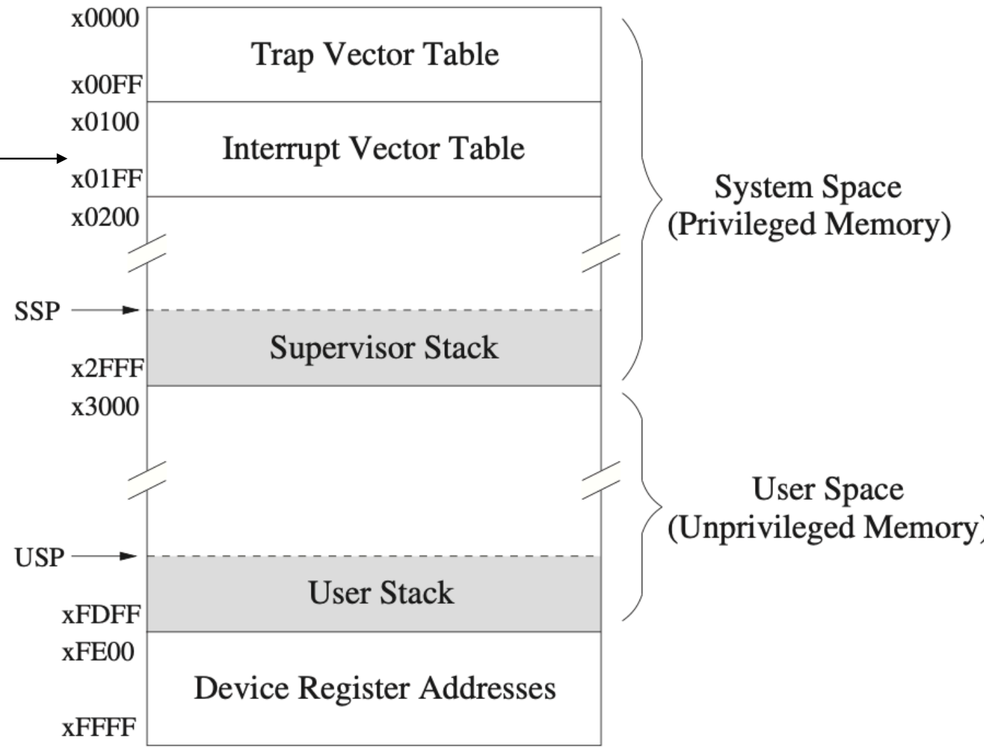
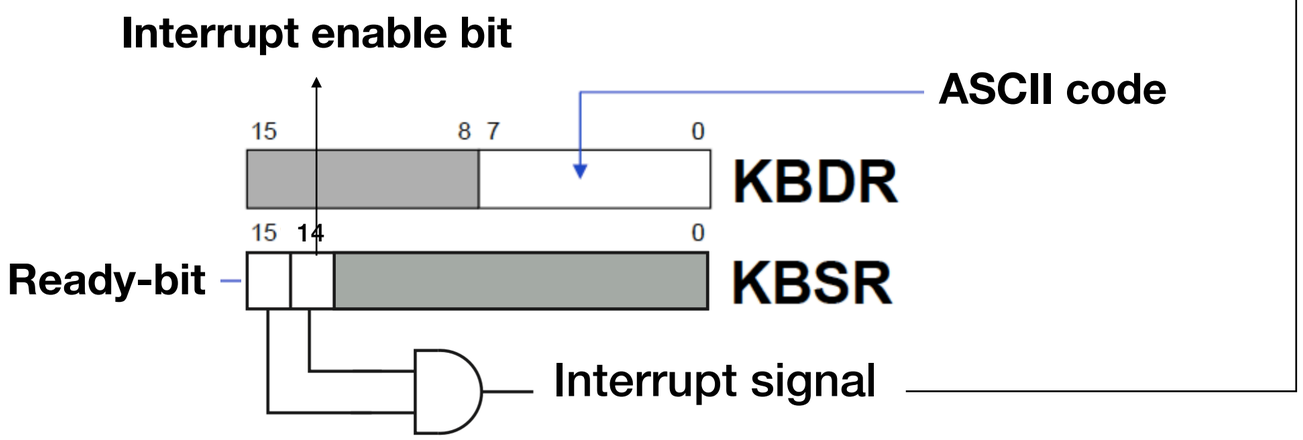


Figure A.1 - P&P 3rd Ed.

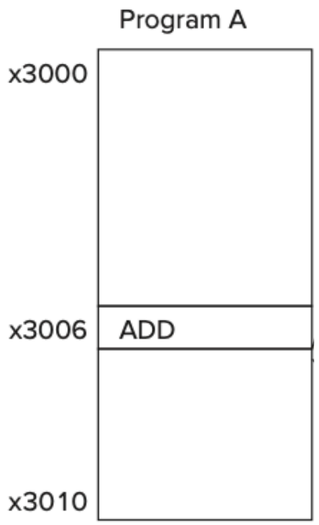


Interrupt example

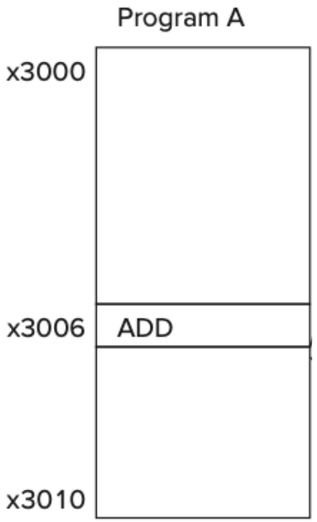
```
.ORIG   x3000
        LEA     R0, ISR_KB
        STI     R0, KBINTV  ; load ISR address to INTV
        LD      R3, EN_IE
        STI     R3, KBSR    ; set IE bit of KBSR
AGAIN   LD      R0, NUM2
        OUT
        BRnzp   AGAIN
ISR_KB  ST      R0, SaveR0  ; callee-save R0
        LDI     R0, KBDR    ; read a char from KB and clear ready bit
        OUT
        LD      R0, SaveR0  ; callee-restore R0
        HALT
;
EN_IE   .FILL   x4000 ; To enable the IE bit
NUM2    .FILL   x0032 ; ASCII Code for '2'
KBSR    .FILL   xFE00
KBDR    .FILL   xFE02
KBINTV  .FILL   x0180 ; INT vector table address for keyboard
SaveR0  .BLKW   #1
.END
```

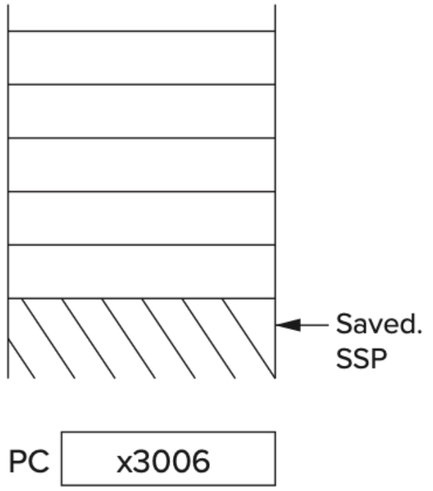
What does this program do?

Interrupting interrupts …



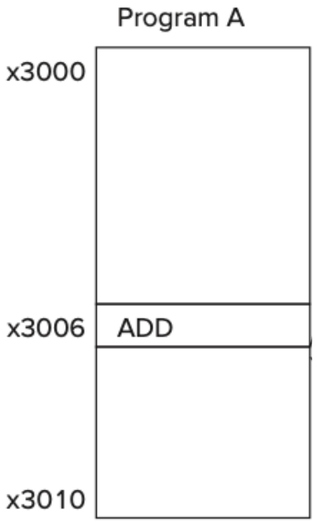
Interrupting interrupts …

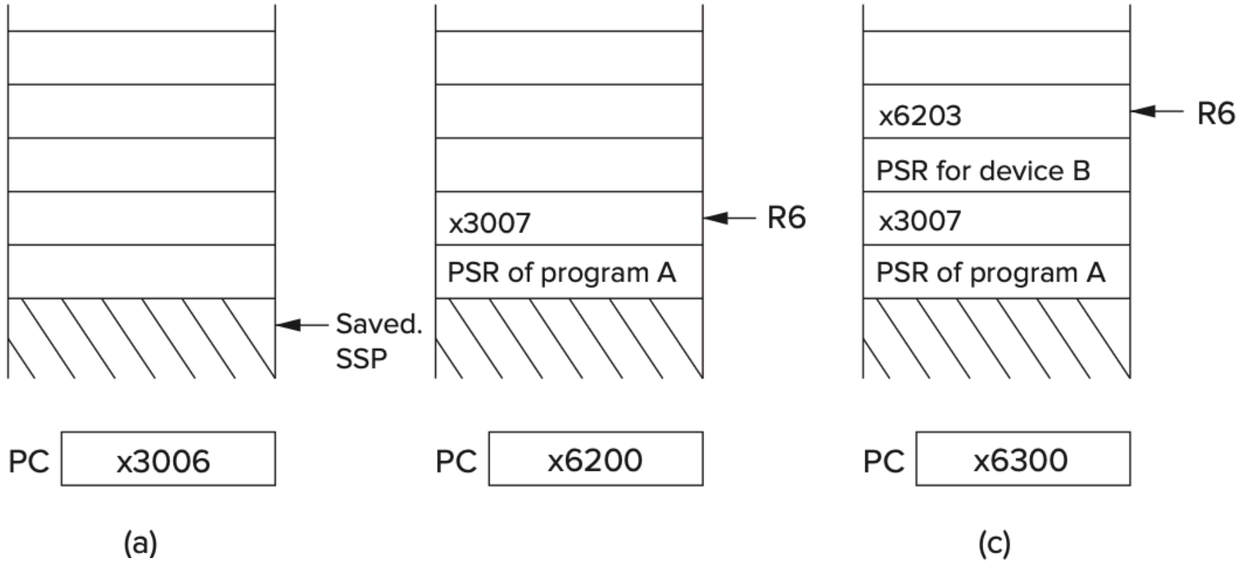




(a)

Interrupting interrupts …





Important addendums, etc.

- C++ is a massive language, we have barely scratched the surface
- E.g. Structs can also have constructors (and inheritance)
  - Often make life easier.

- Consider the following binary tree node definition.

```
struct node{
  int data;
  struct node *left;
  struct node *right;
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

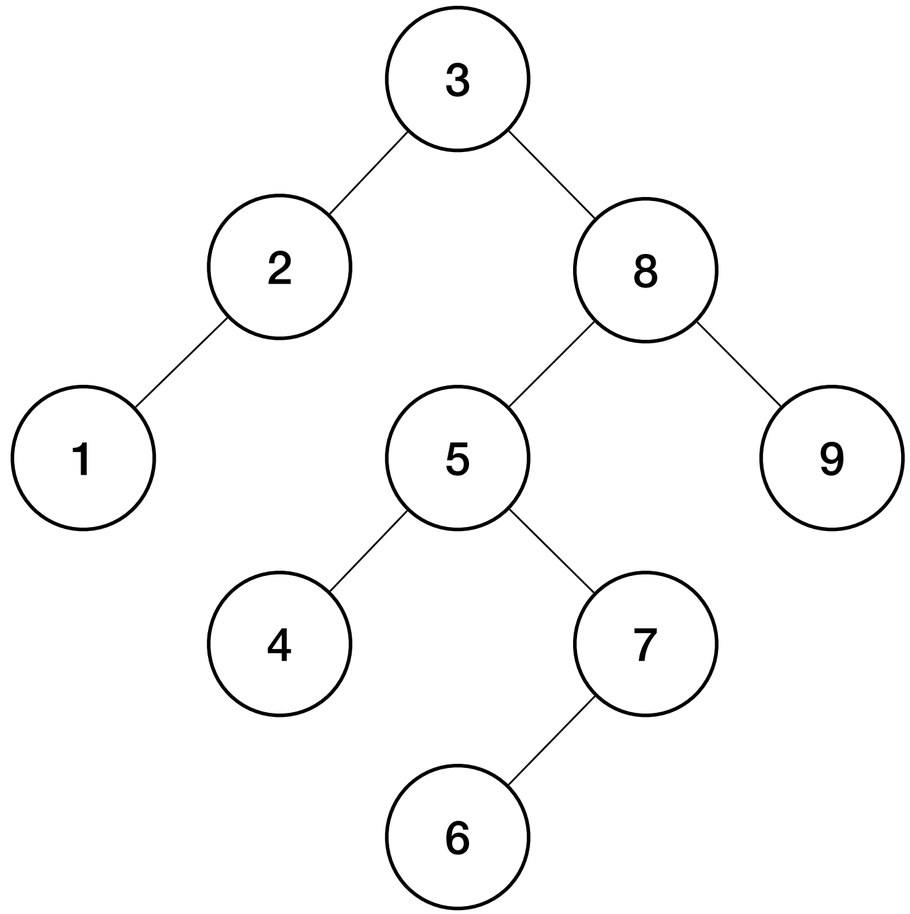
Overloaded constructors

***Initializer list* syntax**

```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

Addendums, etc.

How can we construct this tree using previous slides node definition?

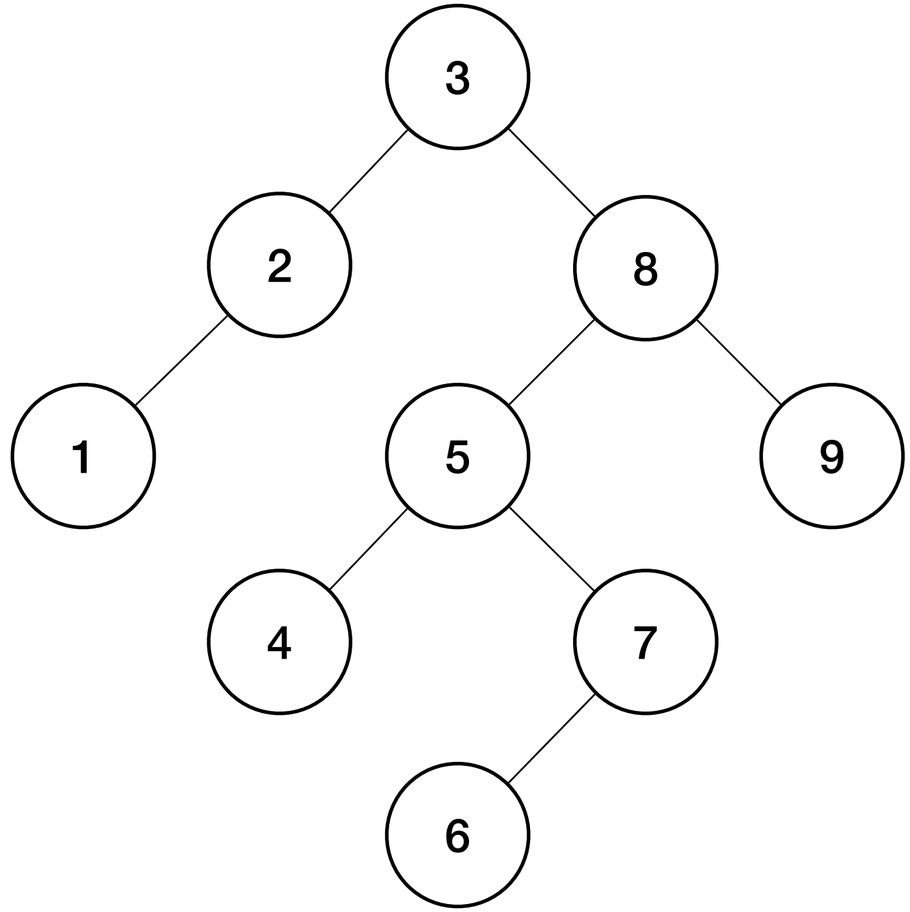


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

Addendums, etc.

How can we construct this tree using previous slides node definition?

```
int main(){
  node *left, *right;

  left = new node(6);
  right = new node(7, left, NULL);
  left = new node(4);
  left = new node(5, left, right);

  right = new node(9);
  right = new node(8, left, right);

  left = new node(1);
  left = new node(2, left, NULL);
  node *root = new node(3, left, right);

  tree_print(root, 0);
}
```

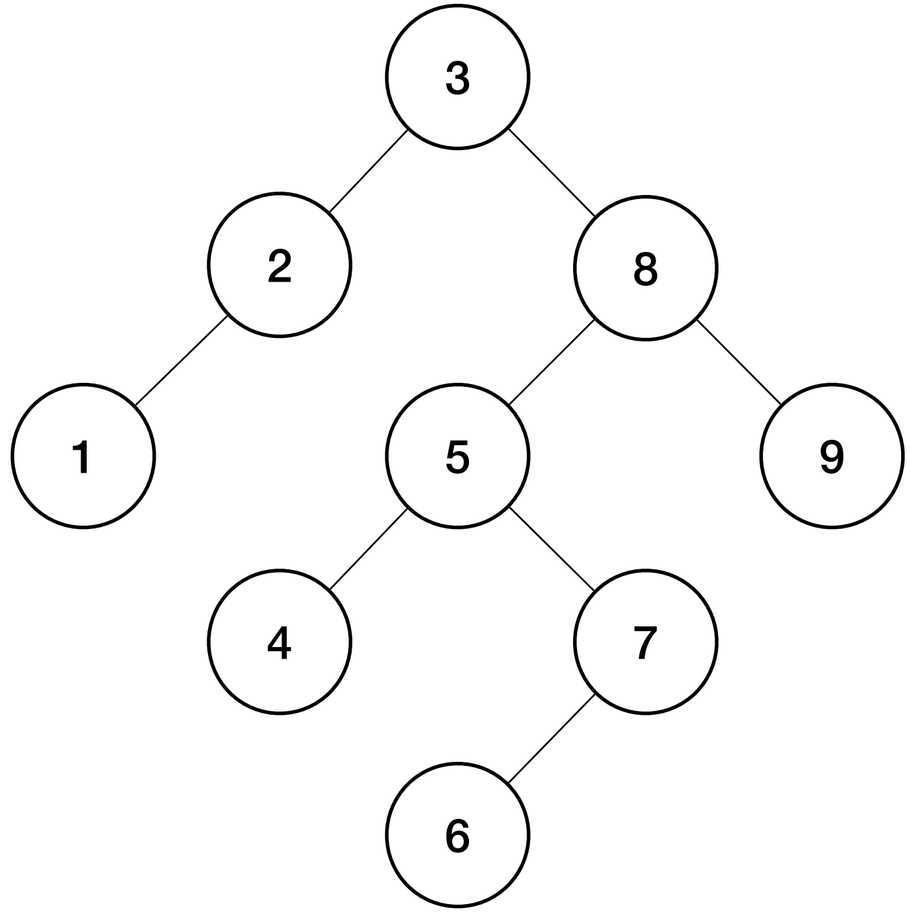


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

Addendums, etc.

How can we construct this tree using previous slides node definition?

```
int main(){
  node *left, *right;

  left = new node(6);
  right = new node(7, left, NULL);
  left = new node(4);
  left = new node(5, left, right);

  right = new node(9);
  right = new node(8, left, right);

  left = new node(1);
  left = new node(2, left, NULL);
  node *root = new node(3, left, right);

  tree_print(root, 0);
}
```



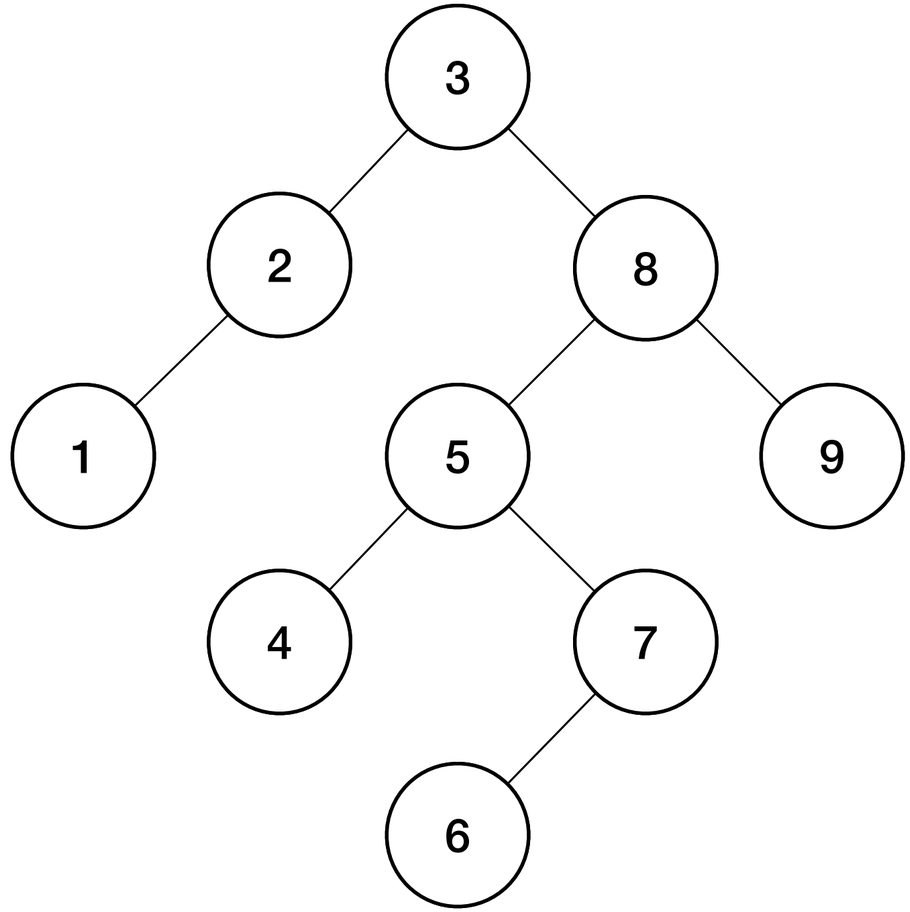
Review exercise: linked list

- Implement a **<u>circularly</u>** linked ***sorted*** list:
  - Print list function
  - Add to list function
    - Add to emtpy list
    - Add in middle
      - Add as first
      - Add as last
- Destroy list function

Review exercises - tree

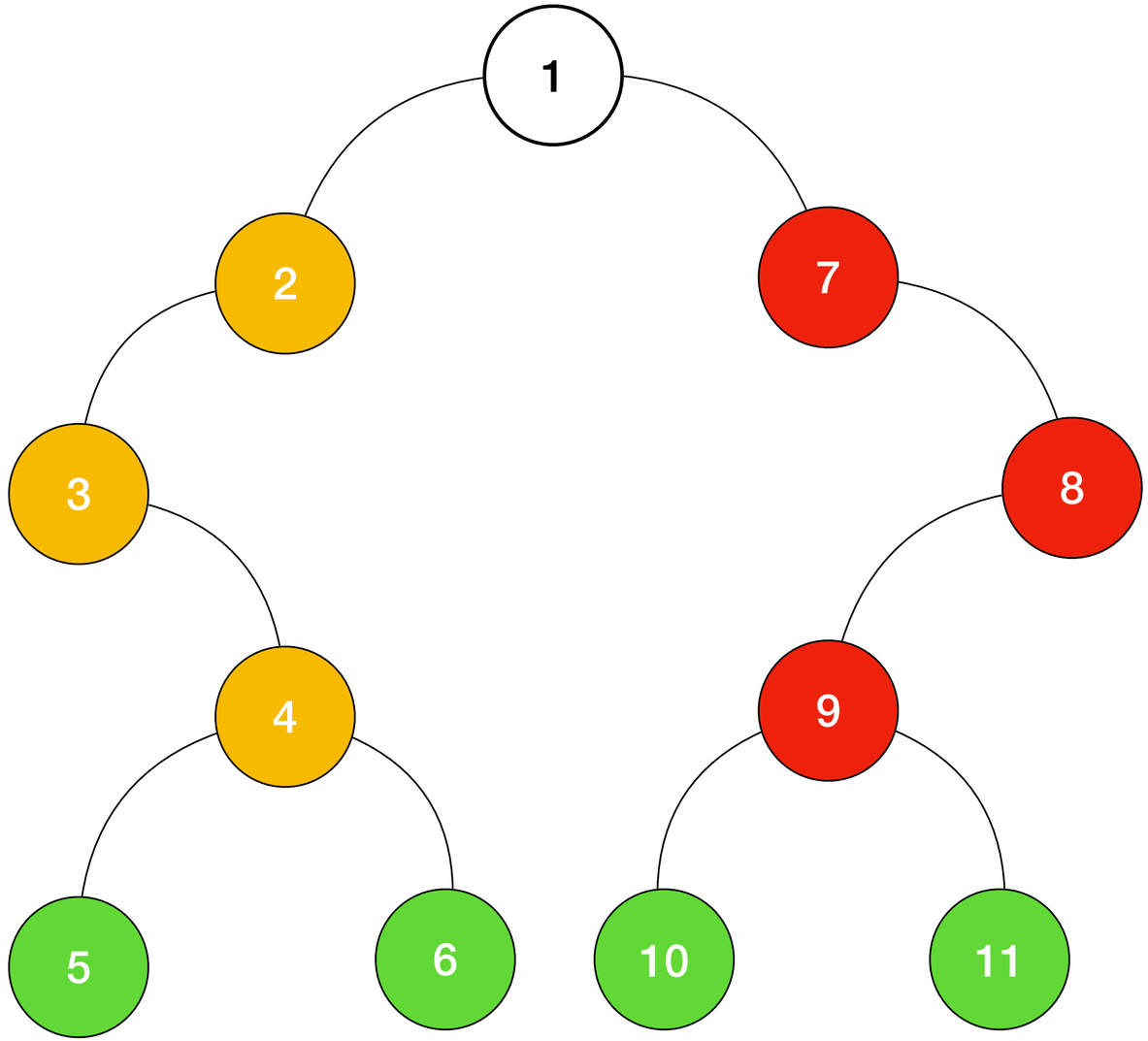
- Given a binary tree print all the boundary nodes starting from the root in counter-clockwise fashion

- Boundary composed of: root +
  - Leaf nodes
  - Left boundary
    - Prefer left if left exists
  - Right boundary
    - Prefer right if right exits
- Should not repeat nodes



Review exercise - tree

- Delineate the left boundary
  - [2, 3, 4]
- Delineate the leaves
  - [5, 6, 10, 11]
- Delineate the right boundary
  - [9, 8, 7]
- Write a function to print all the boundary nodes starting from the root in **counter-clockwise fashion.**



Review exercises

```
void print_left_boundary(node *nd){
  if (nd==NULL)
    return;
  if (nd->left || nd->right)
    cout << nd->data <<", ";
  node *st = nd->left ? nd->left : nd->right;
  print_left_boundary(st);
}
```

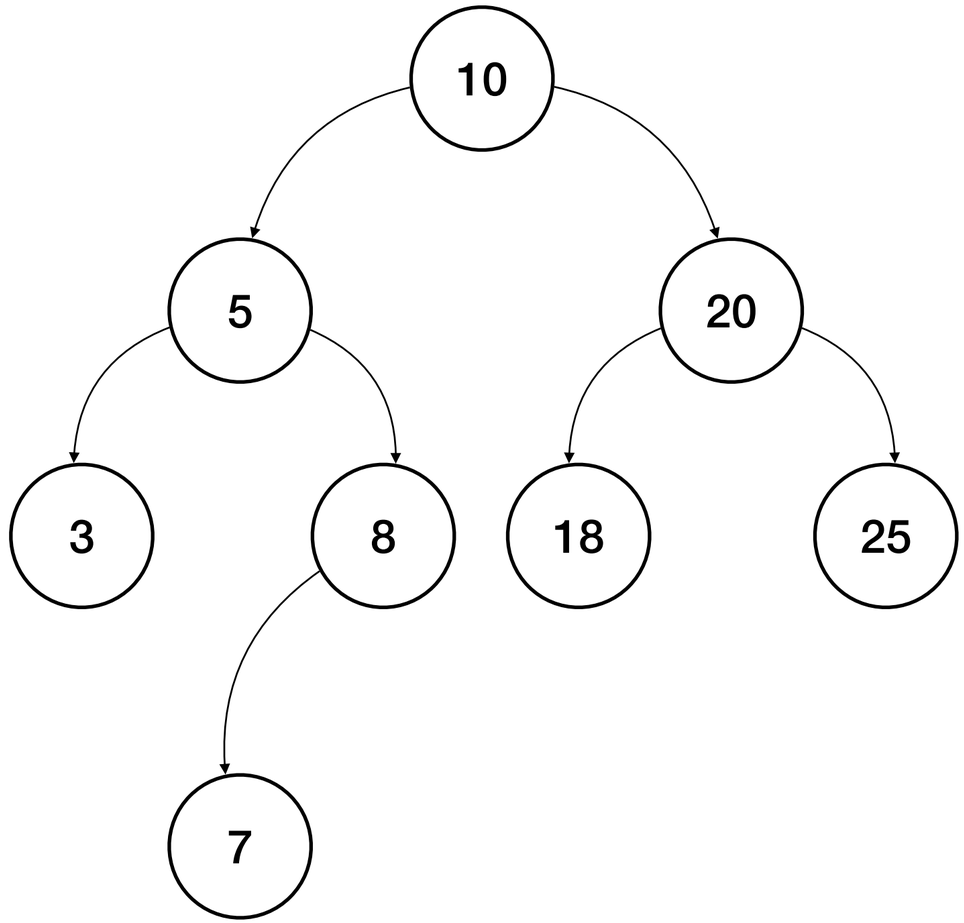
```
void print_leaves(node *cursor){
  if (cursor==NULL)
    return;
  print_leaves(cursor->left);
  if (cursor->left==NULL && cursor->right==NULL)
    cout<<cursor->data<<", ";
  print_leaves(cursor->right);
}
```

```
void print_right_boundary(node *nd){
  if (nd==NULL)
    return;
  node *st = nd->right ? nd->right : nd->left;
  print_right_boundary(st);
  if (nd->left || nd->right)
    cout << nd->data <<", ";
}
```

```
void print_boundary(node *cursor){
    if (cursor == NULL)
        return;
    cout << cursor->data << " ";

    // Print the left boundary
    print_left_boundary(cursor->left);

    // Print all leaf nodes
    print_leaves(cursor->left);
    print_leaves(cursor->right);

    // Print the right boundary
    print_right_boundary(cursor->right);
}
```

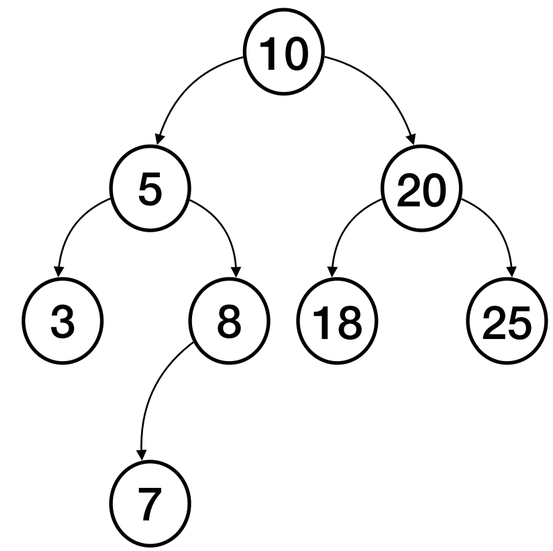
Review exercise - tree

- Is this a BST?
  - Yes
- Write a function to check if given binary tree is a BST
  - Which traversal is helpful?



Review exercise - tree

- Is this a BST?
  - Yes
- Write a function to check if given binary tree is a BST
  - Which traversal is helpful?




```
bool is_bst(node *cursor, node *&prev){
  if (cursor==NULL)
    return true;

  bool left = is_bst(cursor->left, prev);

  if (prev!=NULL && cursor->data <= prev->data)
    return false;

  prev = cursor;
  bool right = is_bst(cursor->right, prev);

  return (left && right);

}
```

Weekend exercise: templates

```
struct node{
  int data;
  struct node *left;
  struct node *right;
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

- Template above definition so it works for Persons!
  - See Github for Person.hpp file

```
typedef node<Person> inode;

int main() {
  inode *left, *right;
  left = new inode(Person("Alice", 65));
  right = new inode(Person("Bob", 67), left, (inode *)nullptr);
  left = new inode(Person("Rick", 68));
  left = new inode(Person("Susan", 65), left, right);

  right = new inode(Person("Mike", 69));
  right = new inode(Person("Matt", 68), left, right);

  left = new inode(Person("Damon", 71));
  left = new inode(Person("Mark", 72), left, (inode *)nullptr);
  inode *root = new inode(Person("Johnny", 73), left, right);

  tree_print(root, 0);
}
```