ECE 220

Lecture x001A - 04/25
Wrap up interrupts & examples

Recap + reminders

Recap + reminders

- MP12 due date has been updated

Recap + reminders

- MP12 due date has been updated

- Final exam on 05/06

Recap + reminders

- MP12 due date has been updated
- Final exam on 05/06
- Conflict exam requests due 05/01

Recap + reminders

- MP12 due date has been updated
- Final exam on 05/06
  - Conflict exam requests due 05/01
- Last day of office hours - 05/01

Recap + reminders

- MP12 due date has been updated
- Final exam on 05/06
  - Conflict exam requests due 05/01
- Last day of office hours - 05/01
- Programming competition

Recap + reminders

- MP12 due date has been updated
- Final exam on 05/06
  - Conflict exam requests due 05/01
- Last day of office hours - 05/01
- Programming competition
- ICES forms now available online

Recap + reminders

- MP12 due date has been updated
- Final exam on 05/06
  - Conflict exam requests due 05/01
- Last day of office hours - 05/01
- Programming competition
- ICES forms now available online
  - 1% extra course grade for 70% across all three sections

Recap + reminders

- MP12 due date has been updated
- Final exam on 05/06
  - Conflict exam requests due 05/01
- Last day of office hours - 05/01
- Programming competition
- ICES forms now available online
  - 1% extra course grade for 70% across all three sections
- Extra credit quiz to be available later today

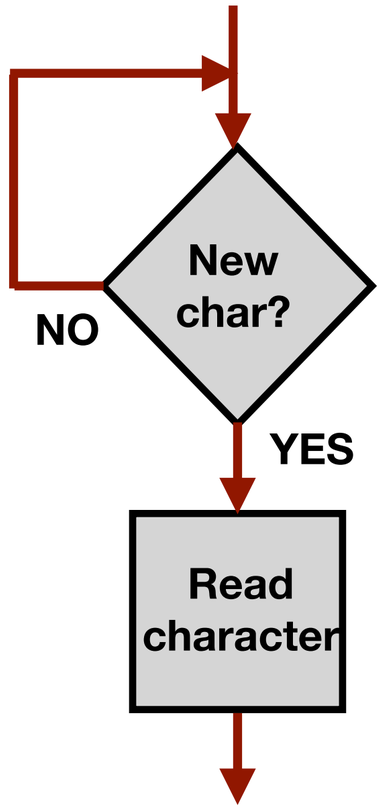
Recap + reminders

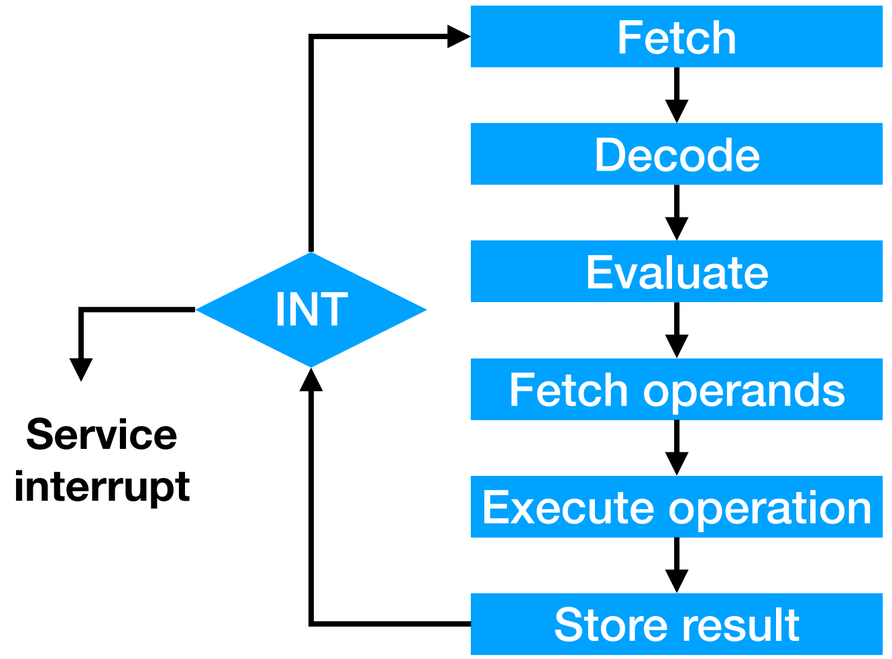
- MP12 due date has been updated
- Final exam on 05/06
  - Conflict exam requests due 05/01
- Last day of office hours - 05/01
- Programming competition
- ICES forms now available online
  - 1% extra course grade for 70% across all three sections
- Extra credit quiz to be available later today
  - Extra credit based on score (35 points)

Recap - Interrupts

Recap - Interrupts

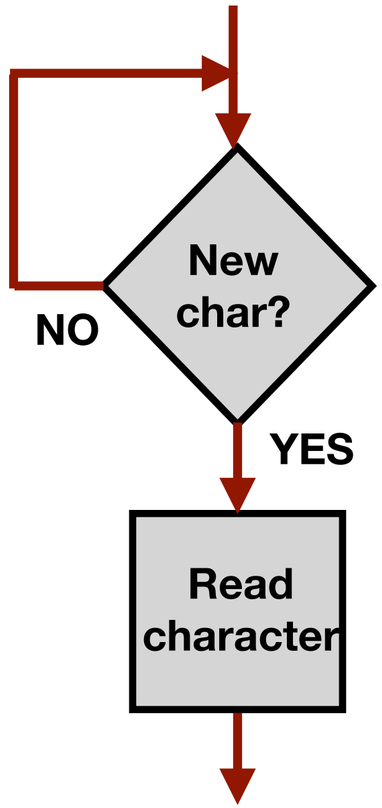
- Polling based I/O vs. Interrupt driven I/O

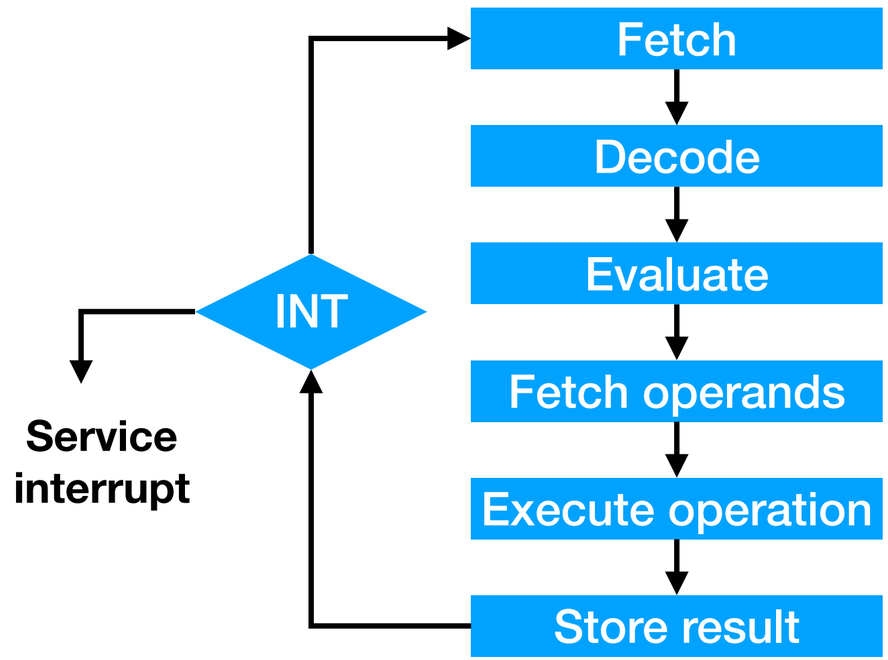




Recap - Interrupts

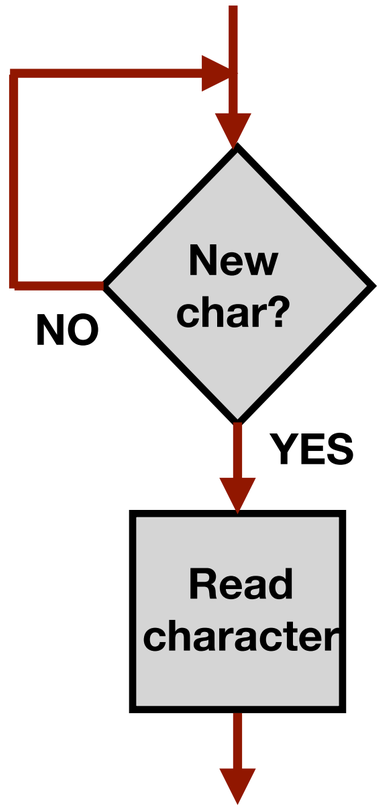
- Polling based I/O vs. Interrupt driven I/O
  - Need to save state of program to be able to resume later

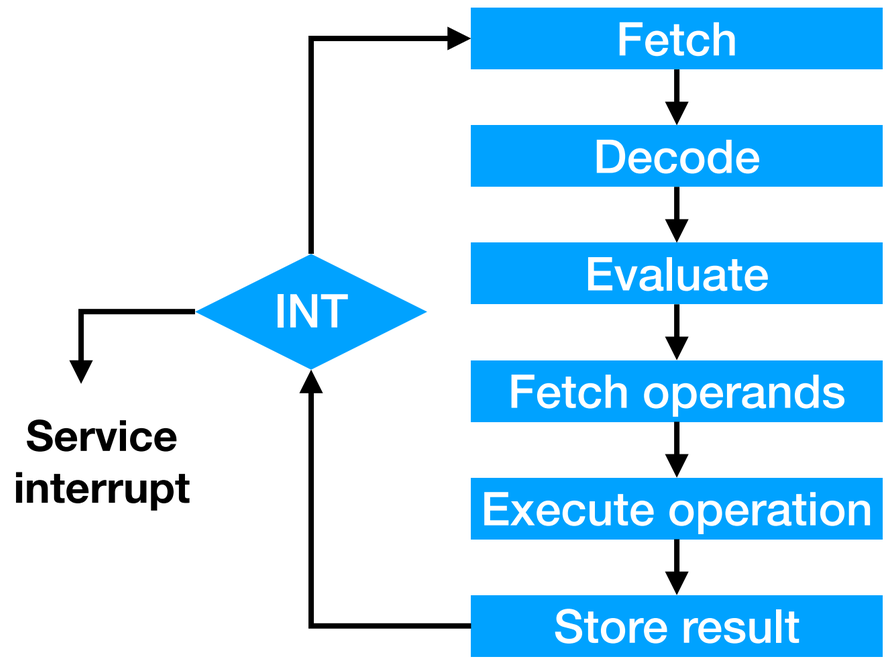




Recap - Interrupts

- 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





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

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

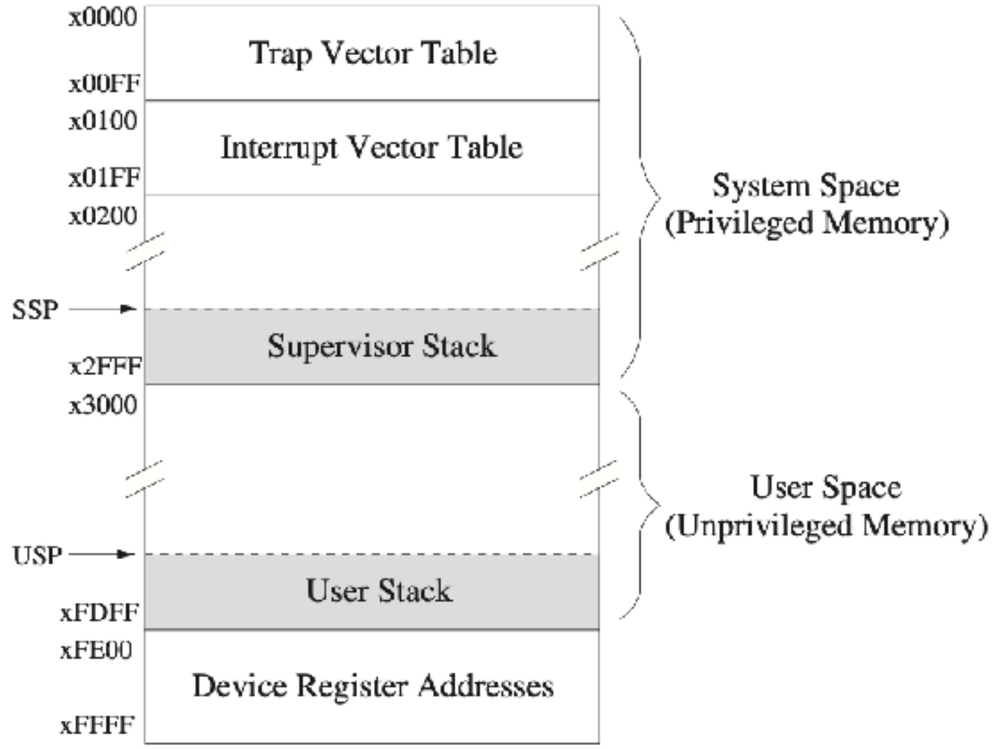


Figure A.1 - P&P 3rd Ed.

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

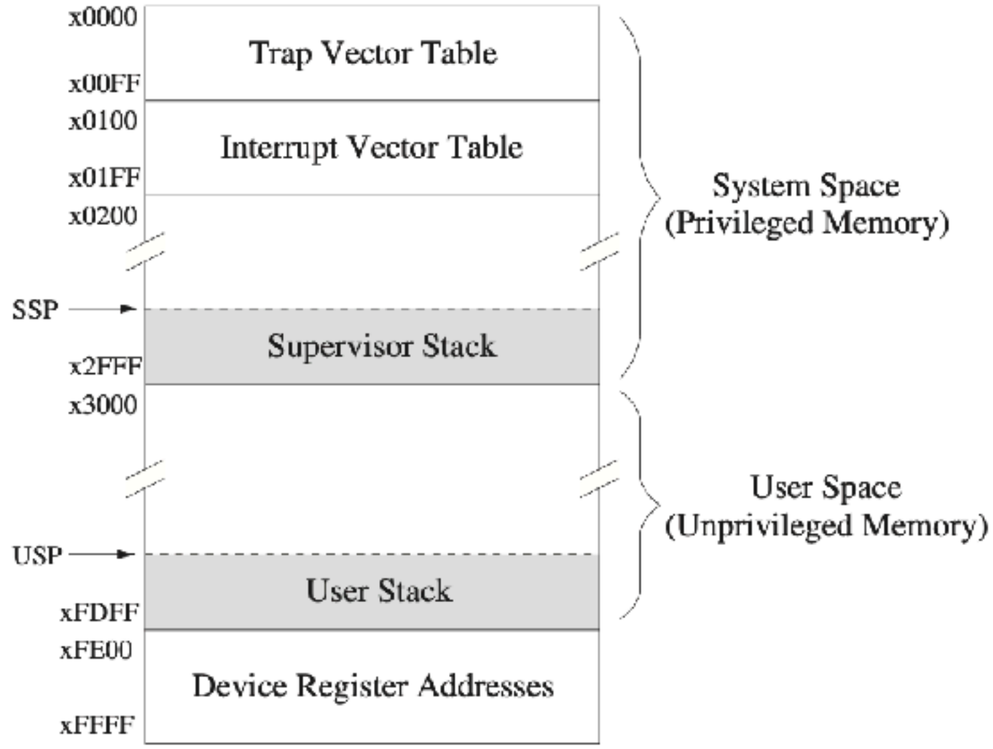


Figure A.1 - P&P 3rd Ed.

Recap - Interrupts

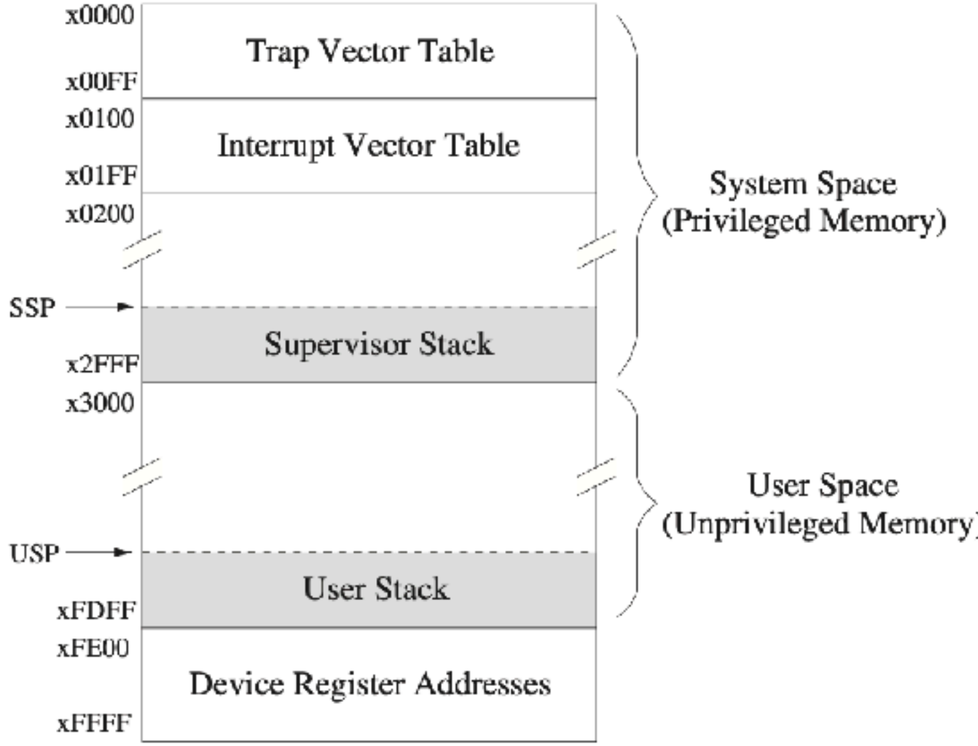
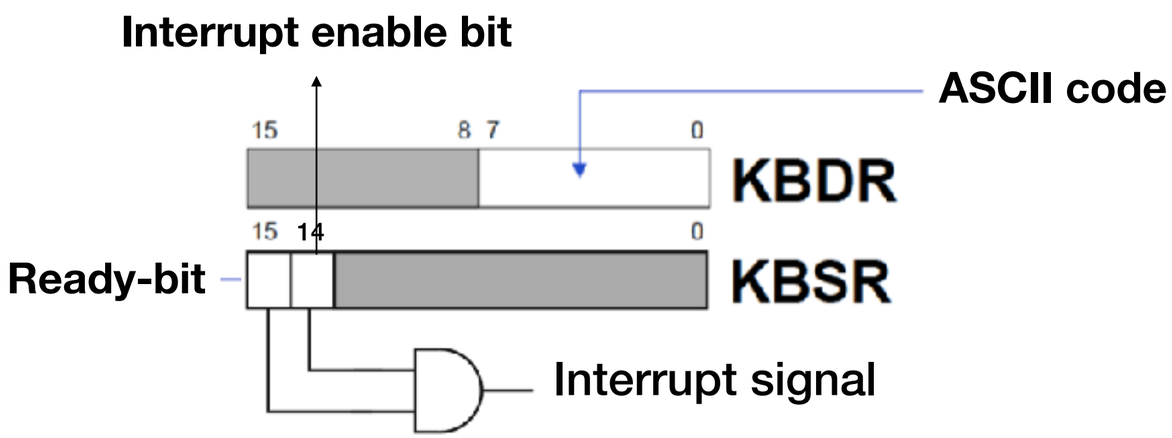
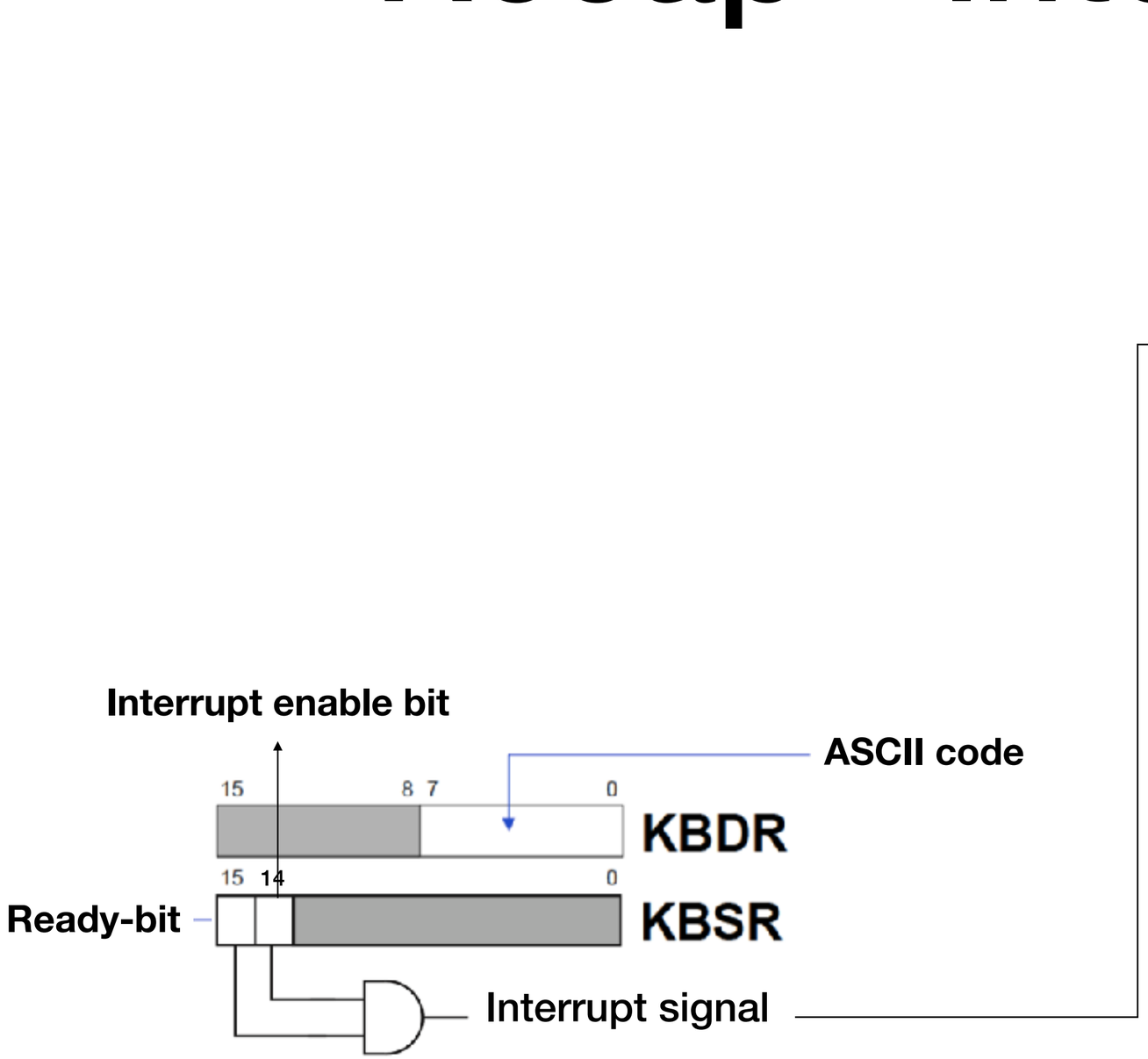


Figure A.1 - P&P 3rd Ed.



Recap - Interrupts



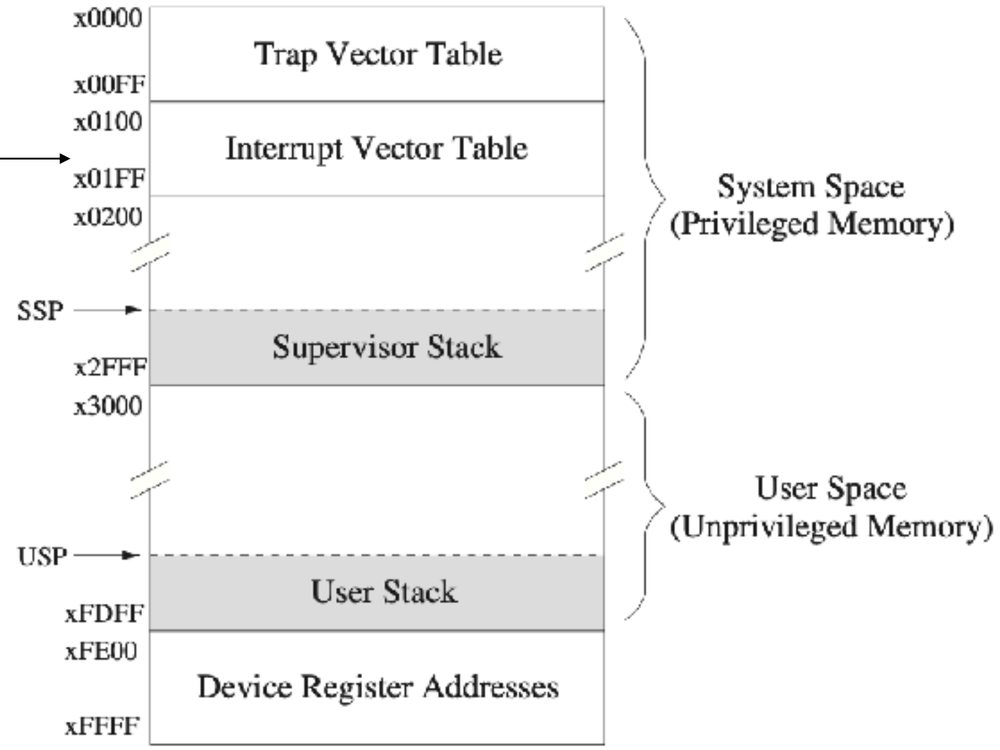


Figure A.1 - P&P 3rd Ed.

Recap - Interrupts

- What needs to be saved?

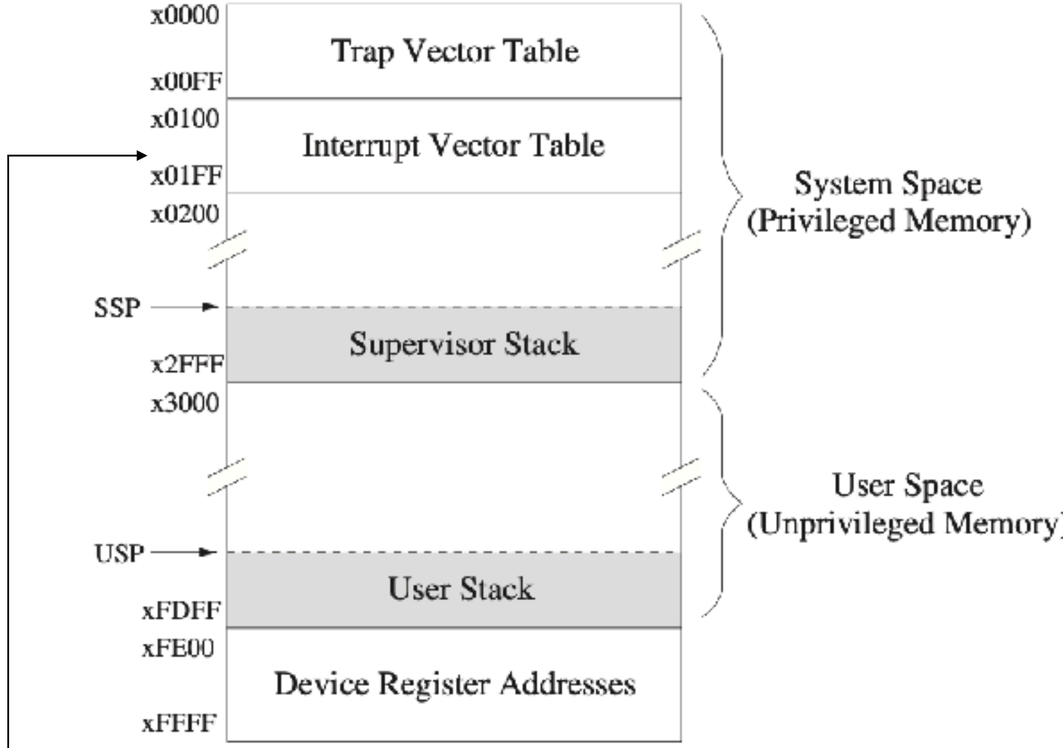
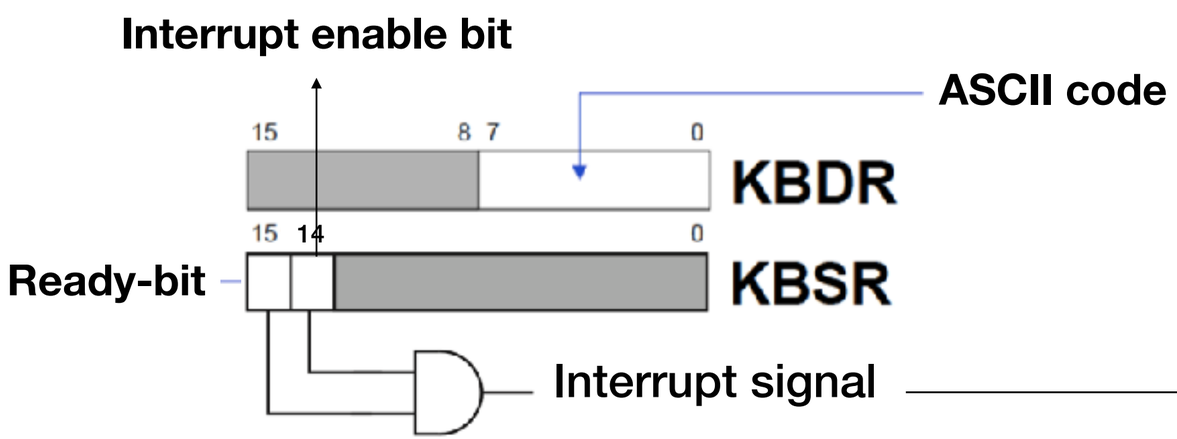


Figure A.1 - P&P 3rd Ed.



Recap - Interrupts

- What needs to be saved?
  - PC, PSR

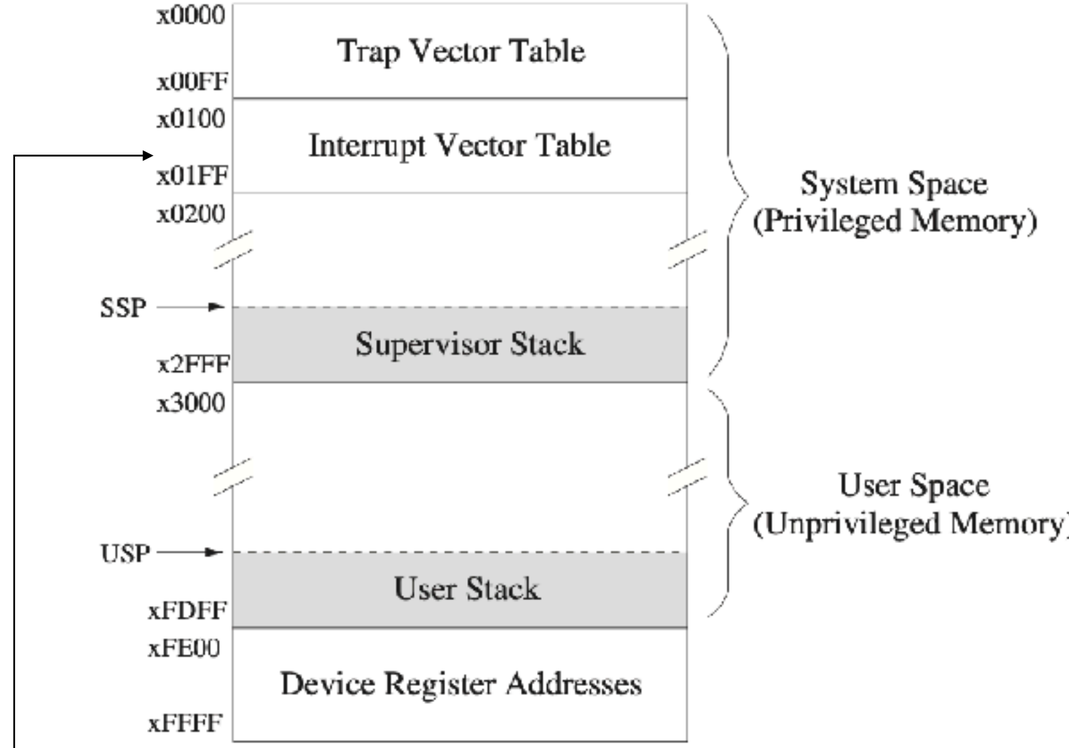
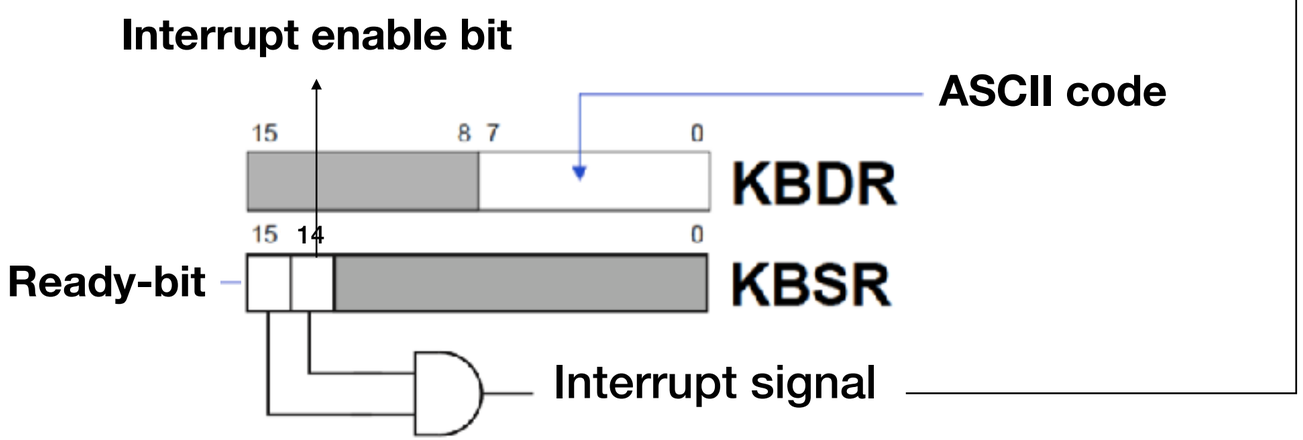
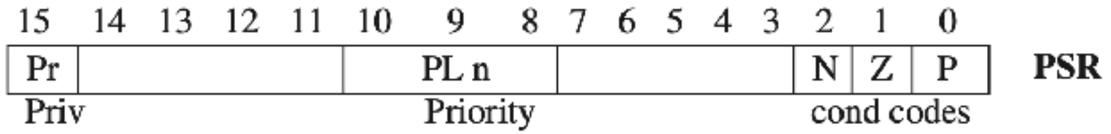


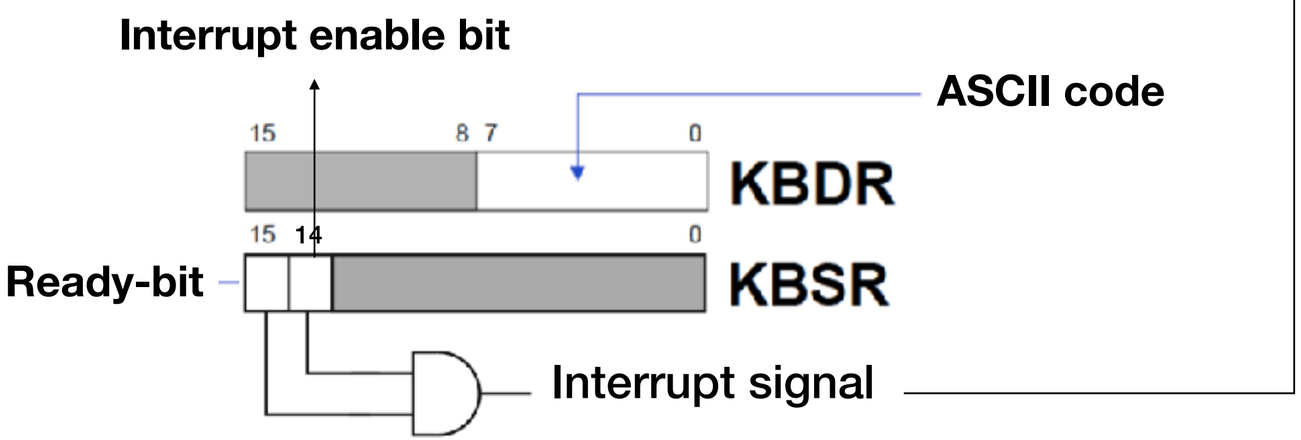
Figure A.1 - P&P 3rd Ed.





Recap - Interrupts

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



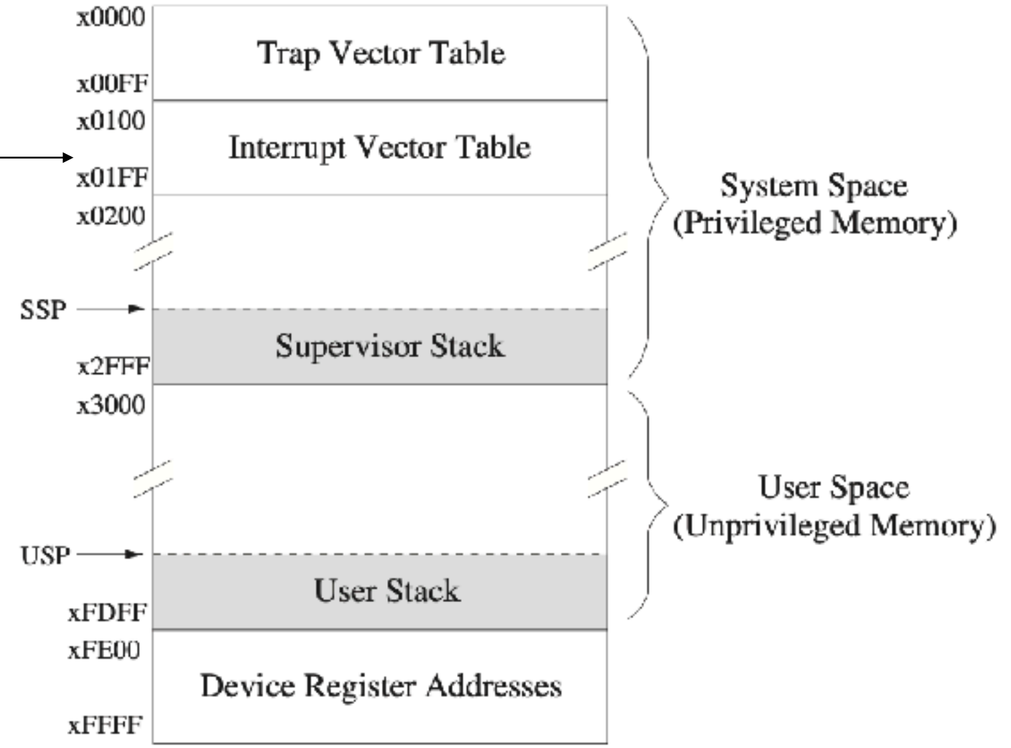


Figure A.1 - P&P 3rd Ed.

https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24/-/tree/main/lec0423

Interrupt example

https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24/-/tree/main/lec0423

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
```

https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24/-/tree/main/lec0423

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?

Important addendums, etc.

Important addendums, etc.

- C++ is a massive language, we have barely scratched the surface

Important addendums, etc.

- C++ is a massive language, we have barely scratched the surface
- E.g. Structs can also have constructors.

Important addendums, etc.

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

Important addendums, etc.

- C++ is a massive language, we have barely scratched the surface
- E.g. Structs can also have constructors.
  - Often make life easier.
  - Consider the following binary tree node definition.

Important addendums, etc.

- C++ is a massive language, we have barely scratched the surface
- E.g. Structs can also have constructors.
  - 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) {};
};
```

Important addendums, etc.

- C++ is a massive language, we have barely scratched the surface
- E.g. Structs can also have constructors.
  - 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

Important addendums, etc.

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

- Consider the following binary tree node definition.

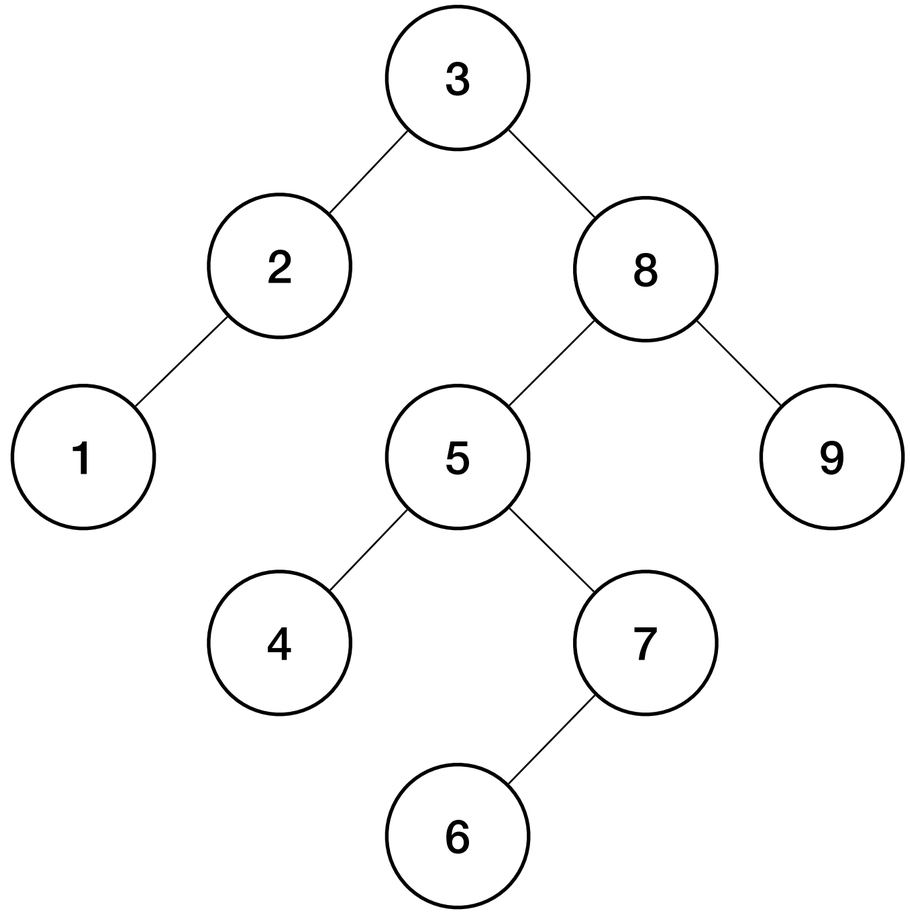
```
struct node{
  int data;
  struct node *left;
  struct node *right;
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

Overloaded constructors

***Initializer list* syntax**

Addendums, etc.

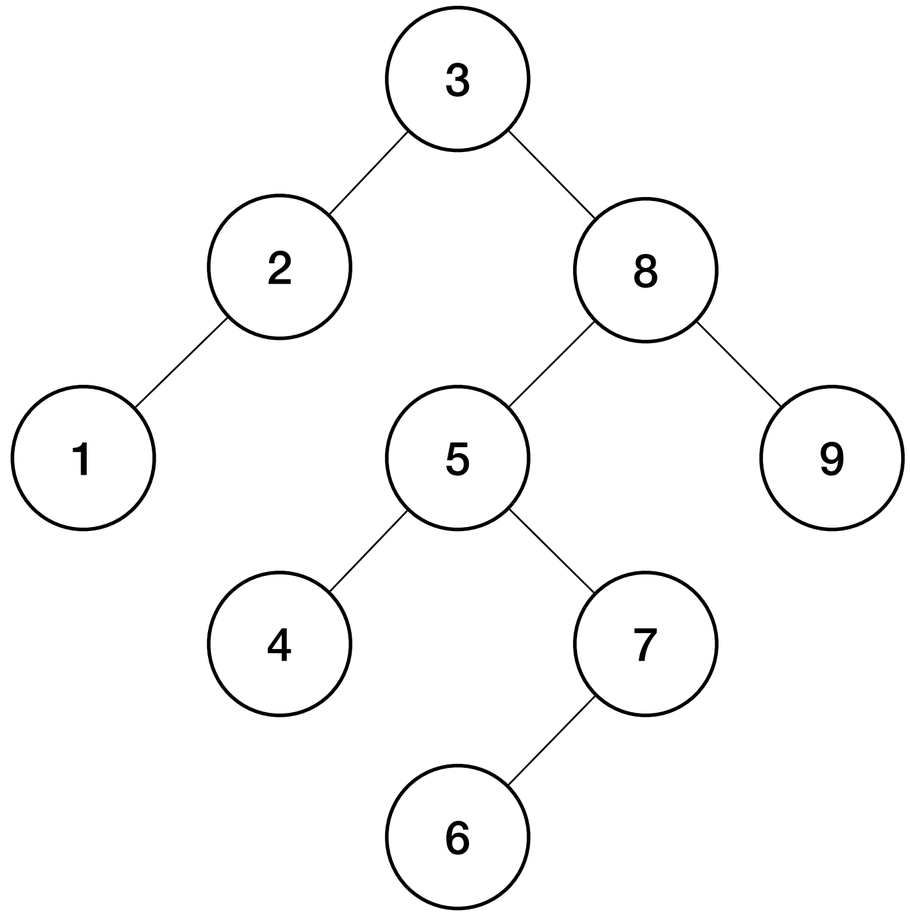
How can we construct this tree using previous slides node definition?



```
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?



```
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?

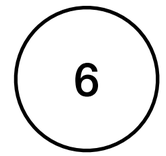
```
int main(){
  node *left, *right;
```

```
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);
```

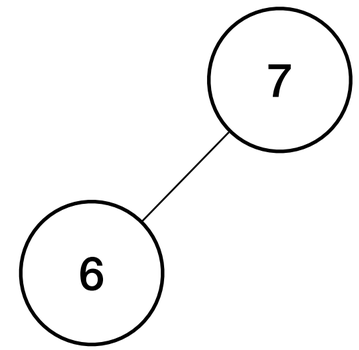


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

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);
```

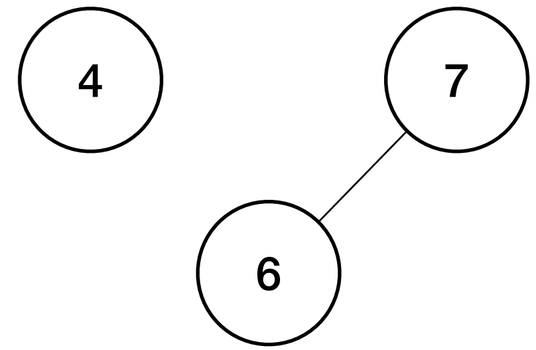


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

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);
```

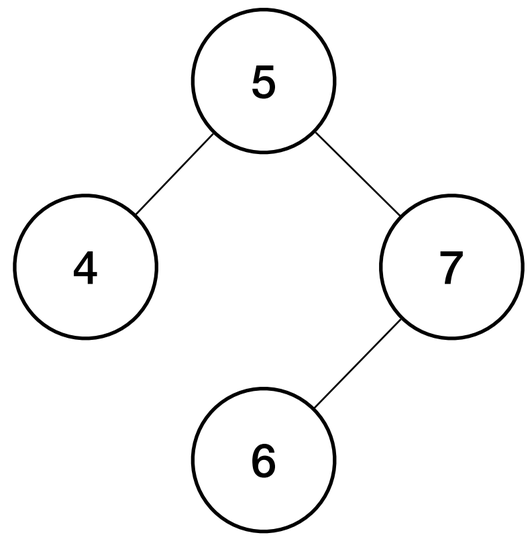


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

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);
```

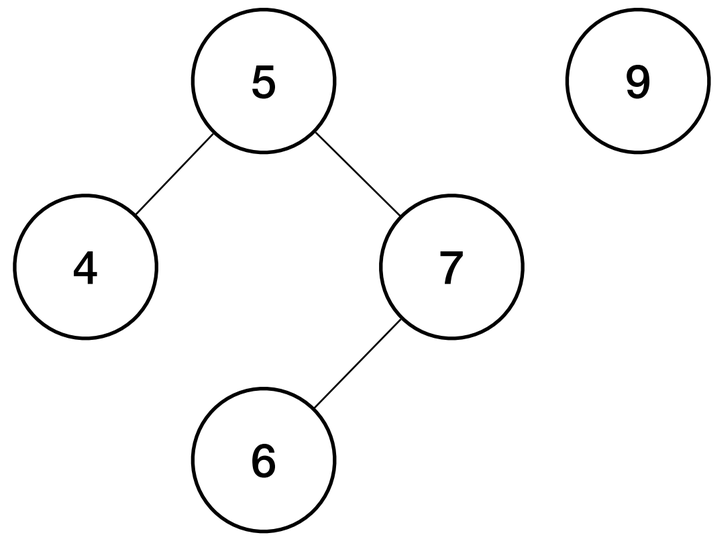


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

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);
```

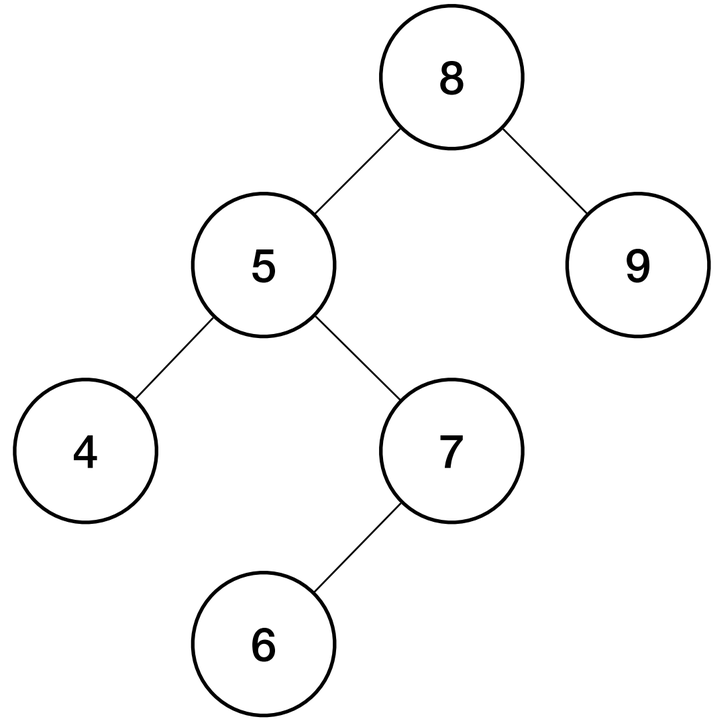


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

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);
```

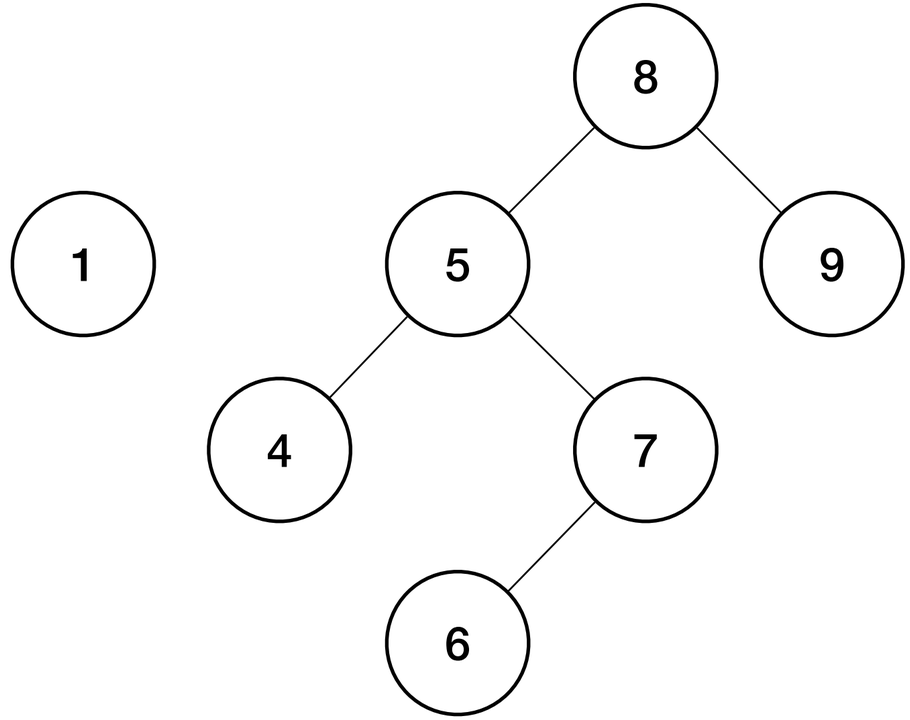


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

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);
```

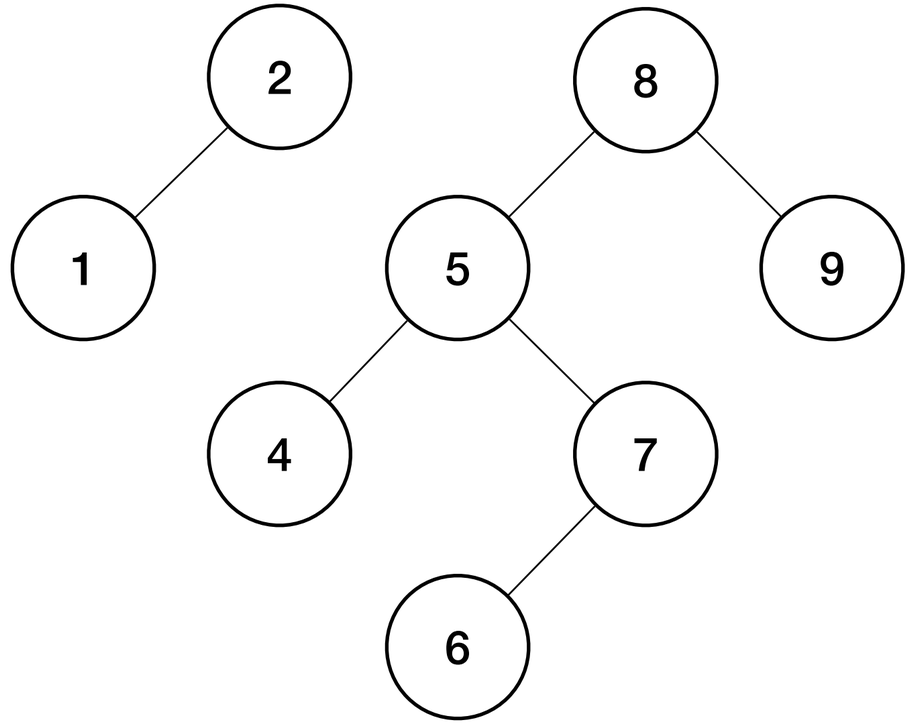


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

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);
```

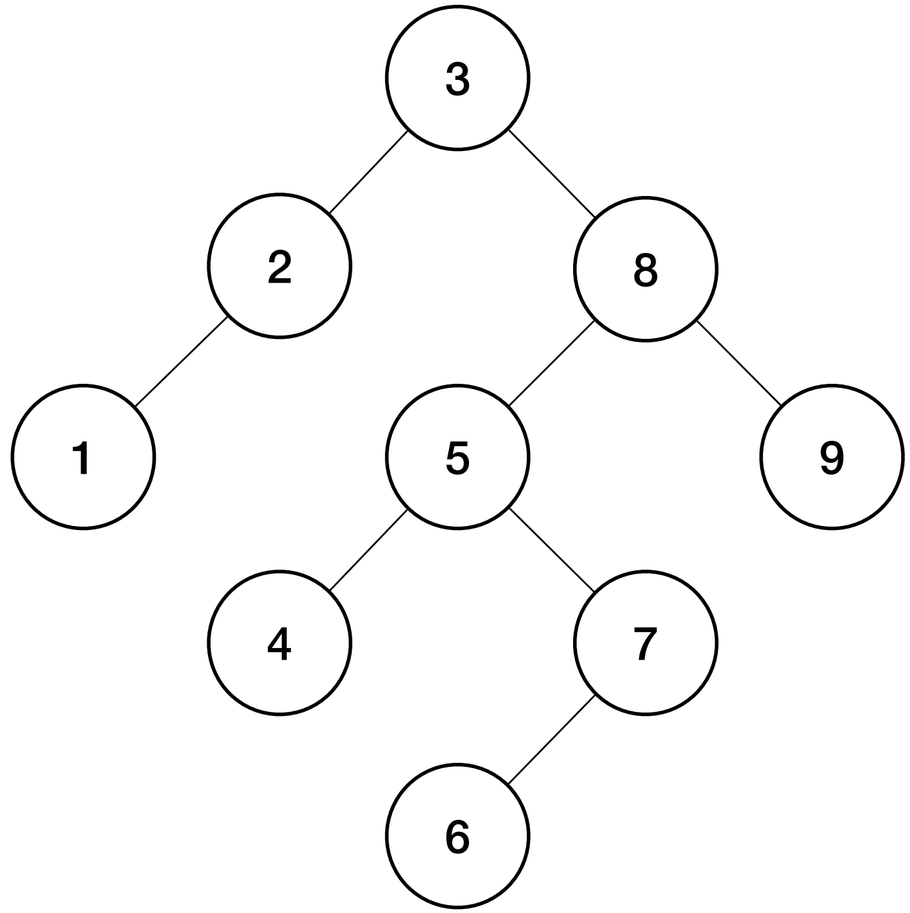


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

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);
```

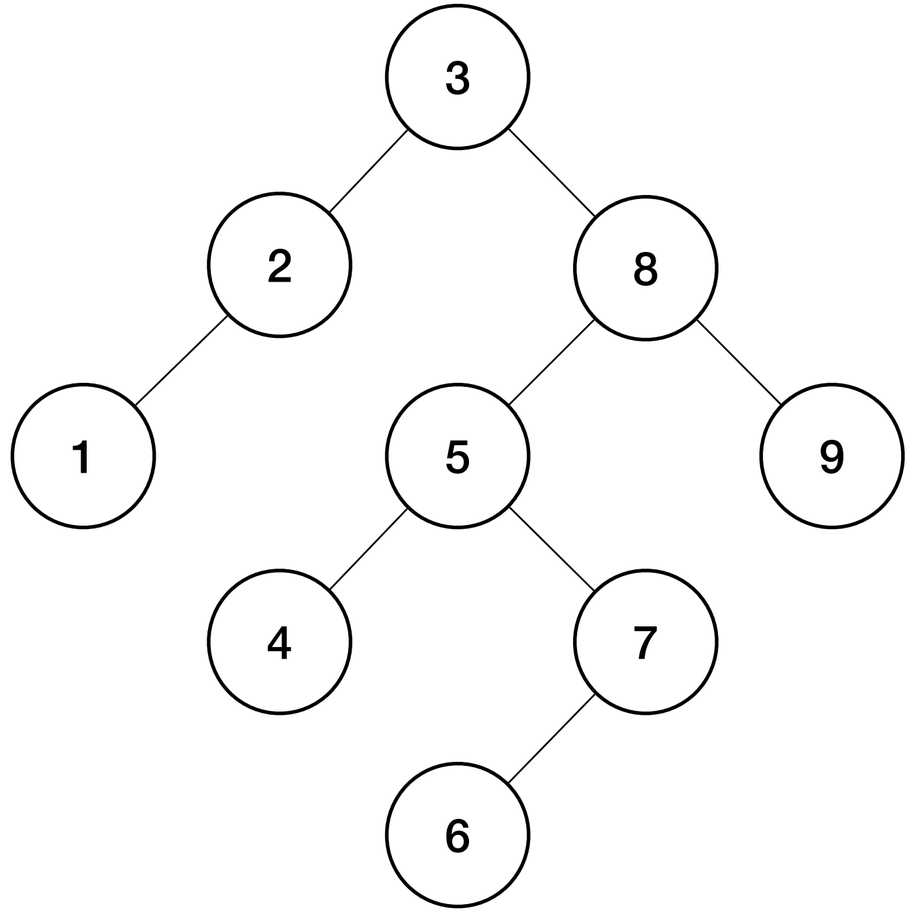


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

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);
}
```

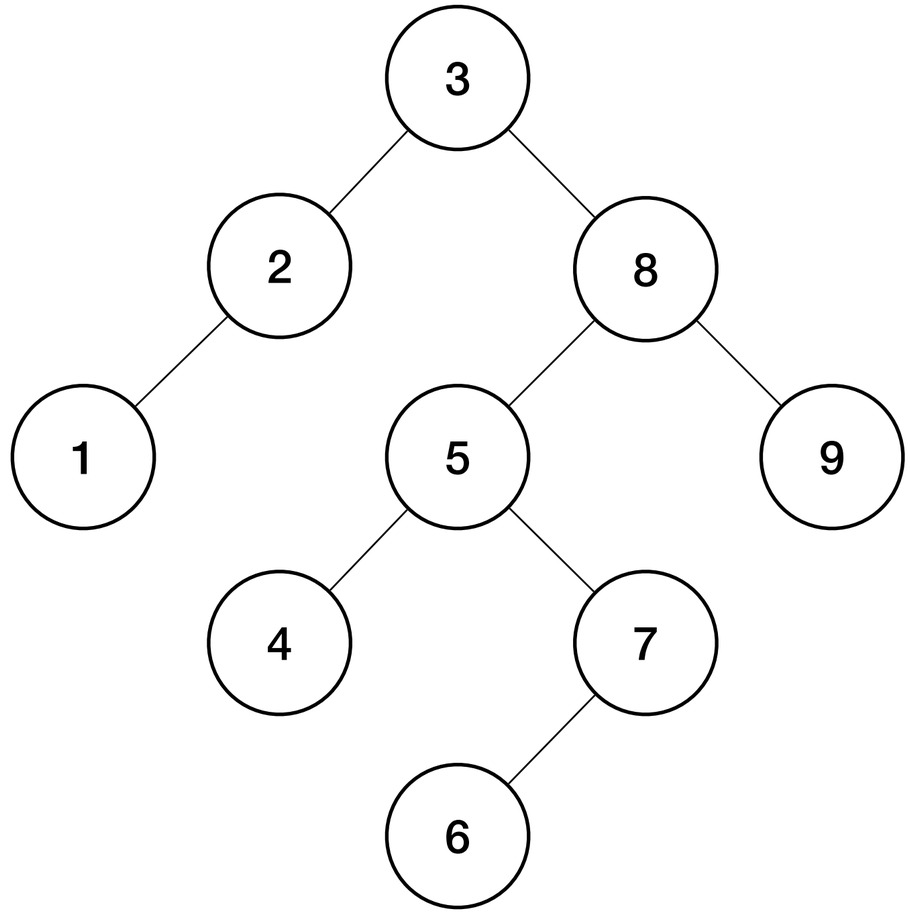


```
struct node{
    …
  node() : data(0), left(NULL), right(NULL) {};
  node(int d): data(d), left(NULL), right(NULL) {};
  node(int d, node *l, node *r): data(d), left(l), right(r) {};
};
```

Addendums, etc.

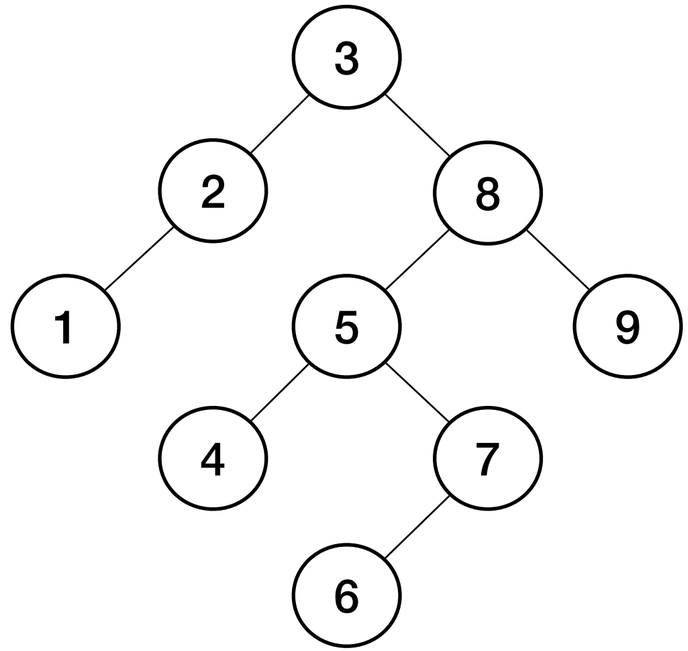
How can we construct this tree using previous slides node definition?

```
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);
}
```



Important addendums, etc.

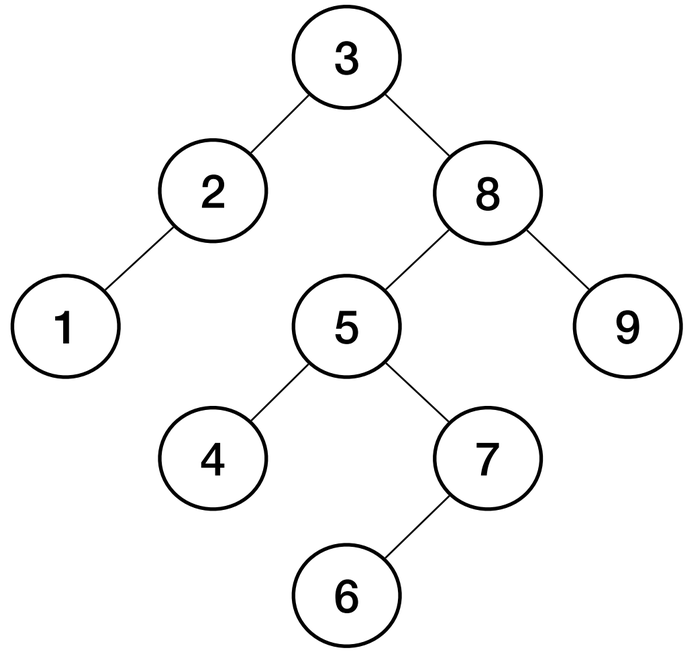
```c
void tree_print(node *cursor, int depth){
    if (cursor == NULL)
      return;
    for (int i = 0; i < depth; i++)
      printf(i == depth - 1 ? "|-" : "  ");
    printf("%d\n", cursor->data);
    tree_print(cursor->left, depth + 1);
    tree_print(cursor->right, depth + 1);
}
```



Important addendums, etc.

Ternary operator
condition ? true : false

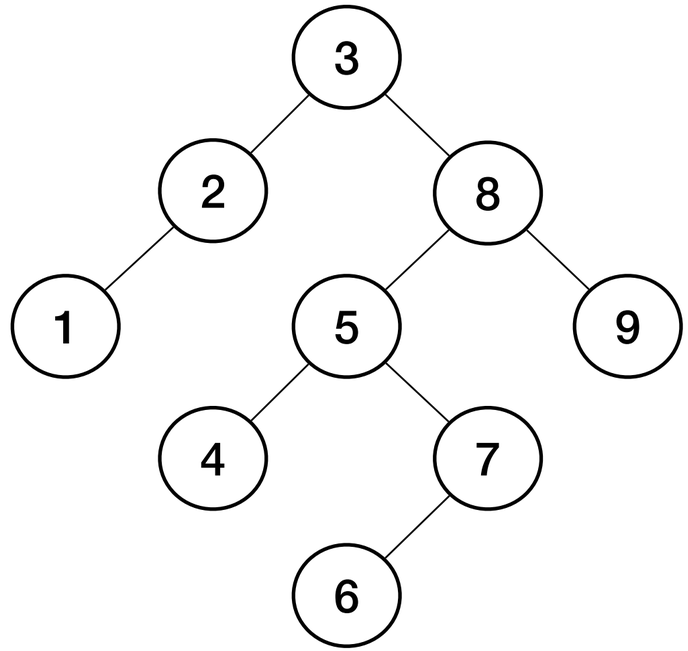
```
void tree_print(node *cursor, int depth){
    if (cursor == NULL)
      return;
    for (int i = 0; i < depth; i++)
      printf(i == depth - 1 ? "|-" : "   ");
    printf("%d\n", cursor->data);
    tree_print(cursor->left, depth + 1);
    tree_print(cursor->right, depth + 1);
}
```



Important addendums, etc.

Ternary operator
condition ? true : false

```
void tree_print(node *cursor, int depth){
    if (cursor == NULL)
      return;
    for (int i = 0; i < depth; i++)
      printf(i == depth - 1 ? "|-" : "  ");
    printf("%d\n", cursor->data);
    tree_print(cursor->left, depth + 1);
    tree_print(cursor->right, depth + 1);
}
```




```
3
|-2
  |-1
|-8
  |-5
    |-4
    |-7
      |-6
|-9
```

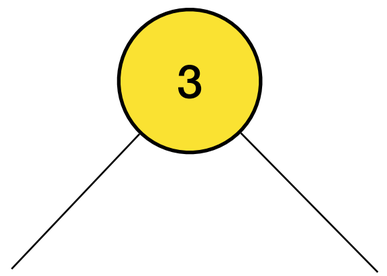
Review exercises - tree

Review exercises - tree

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

Review exercises - tree

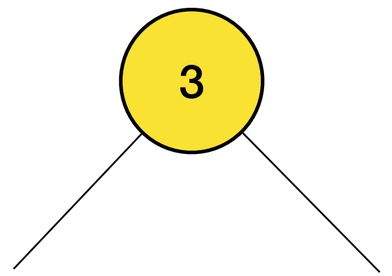
- Given a binary tree print all the boundary nodes starting from the root in counter-clockwise fashion



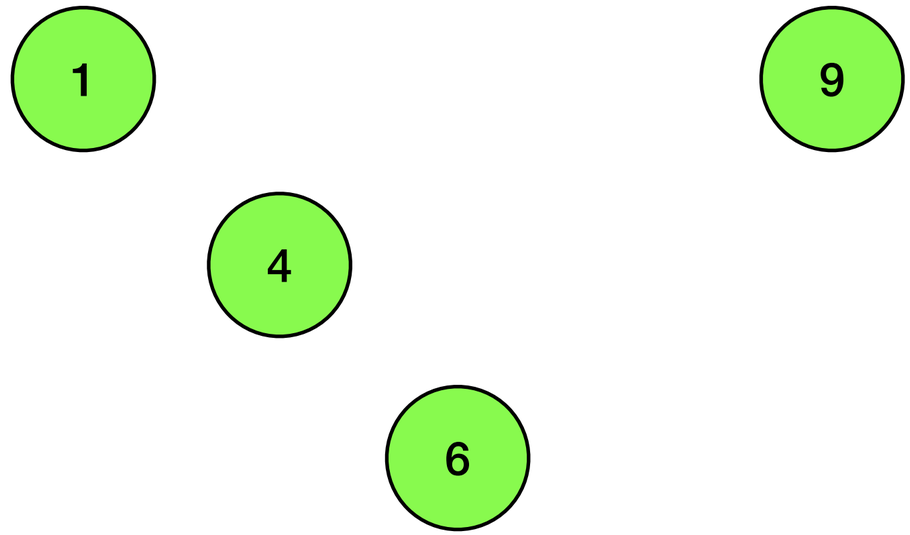
- Boundary composed of: root +

Review exercises - tree

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



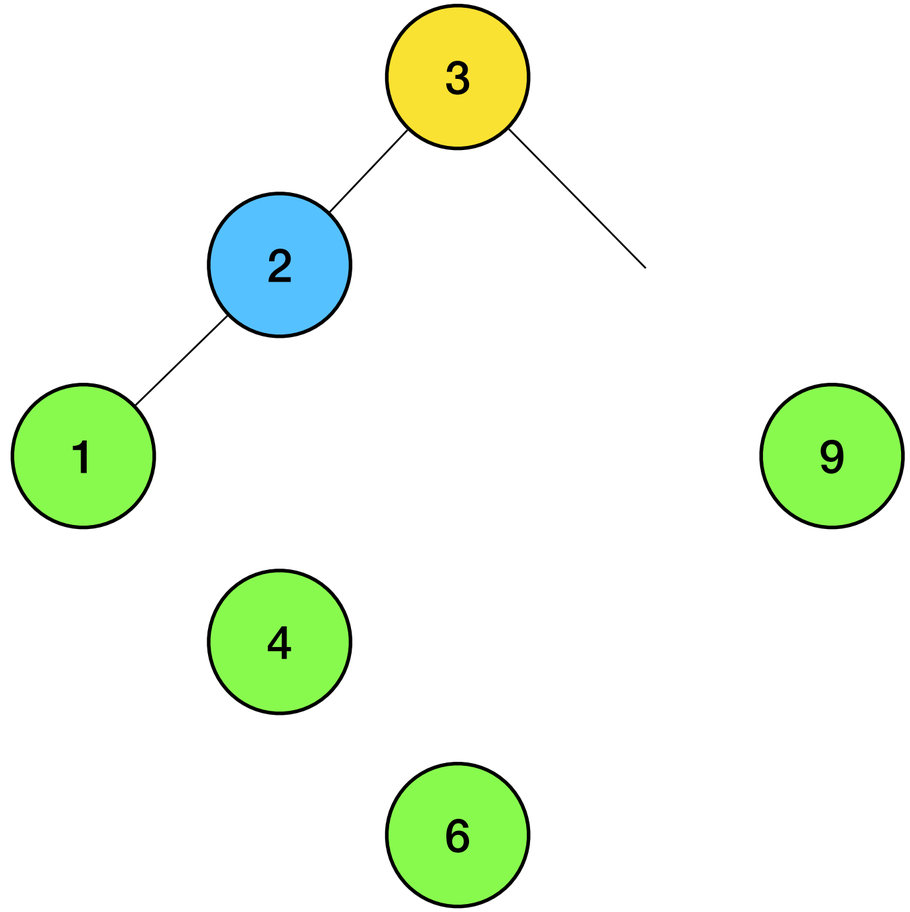
- Boundary composed of: root +
  - Leaf nodes



Review exercises - tree

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

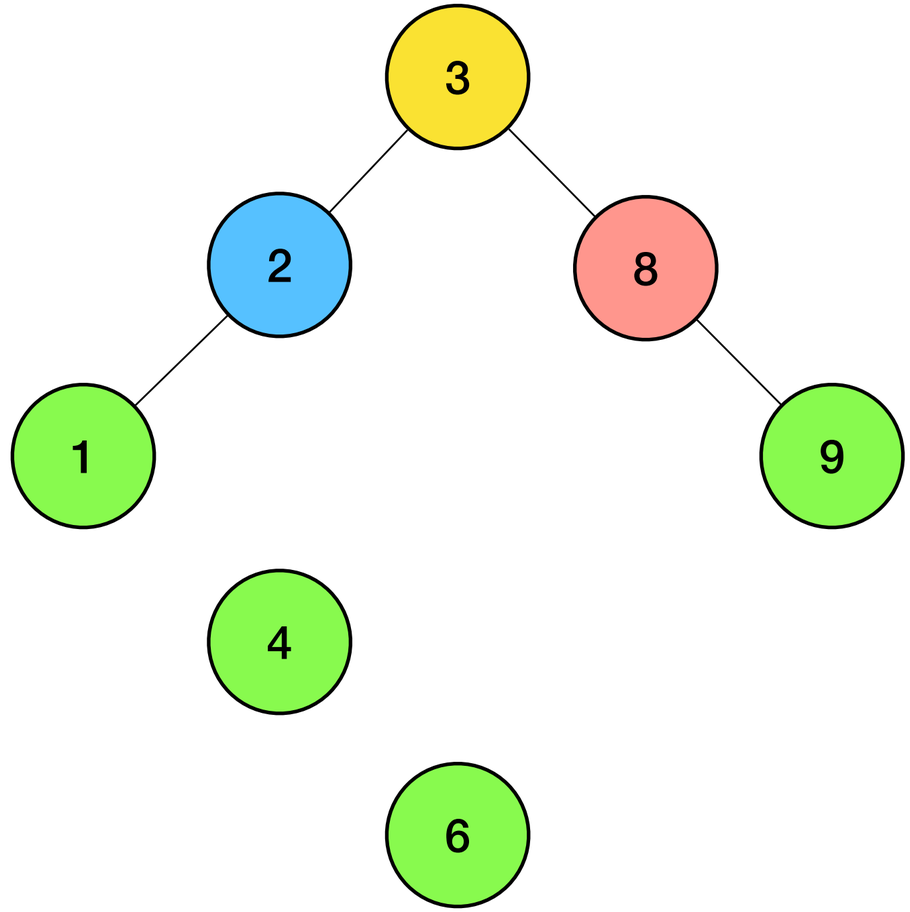
- Boundary composed of: root +
  - Leaf nodes
  - Left boundary



Review exercises - tree

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

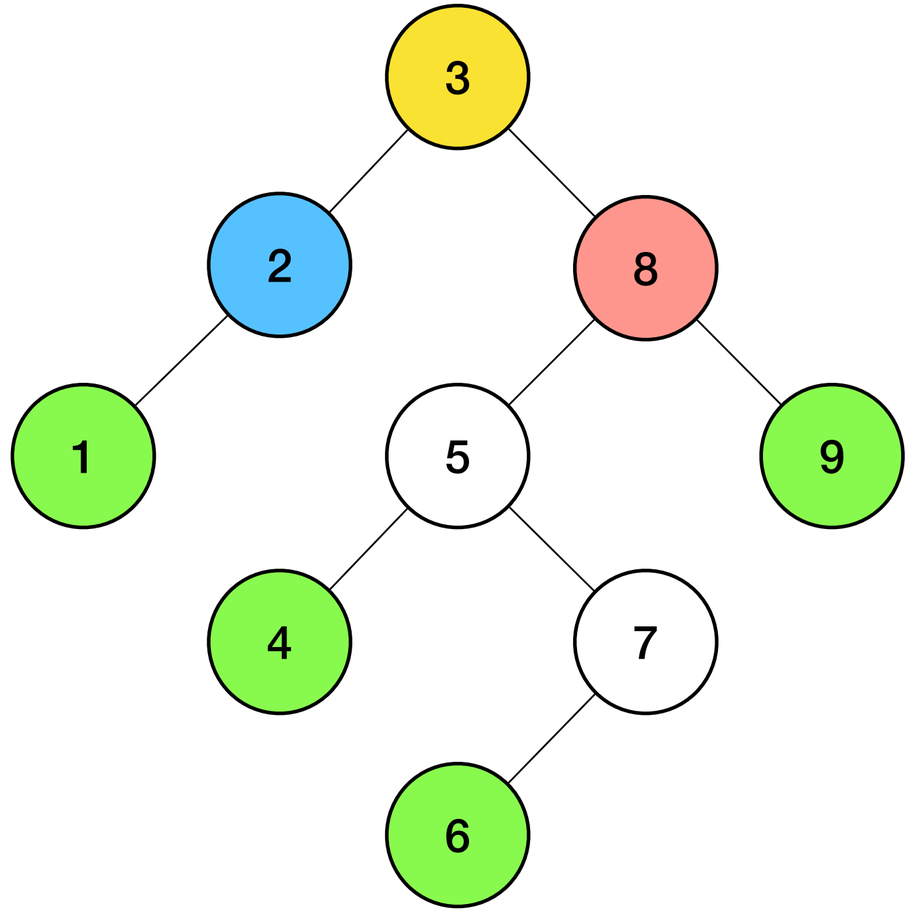
- Boundary composed of: root +
  - Leaf nodes
  - Left boundary
  - Right boundary



Review exercises - tree

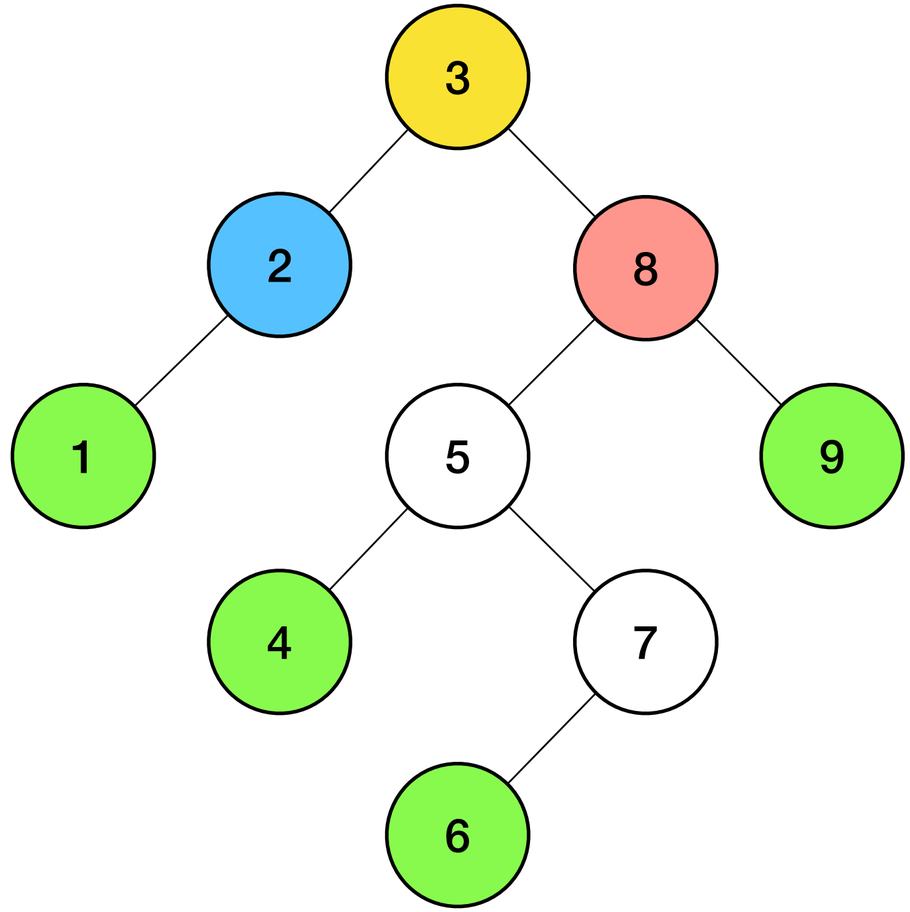
- Given a binary tree print all the boundary nodes starting from the root in counter-clockwise fashion

- Boundary composed of: root +
  - Leaf nodes
  - Left boundary
  - Right boundary

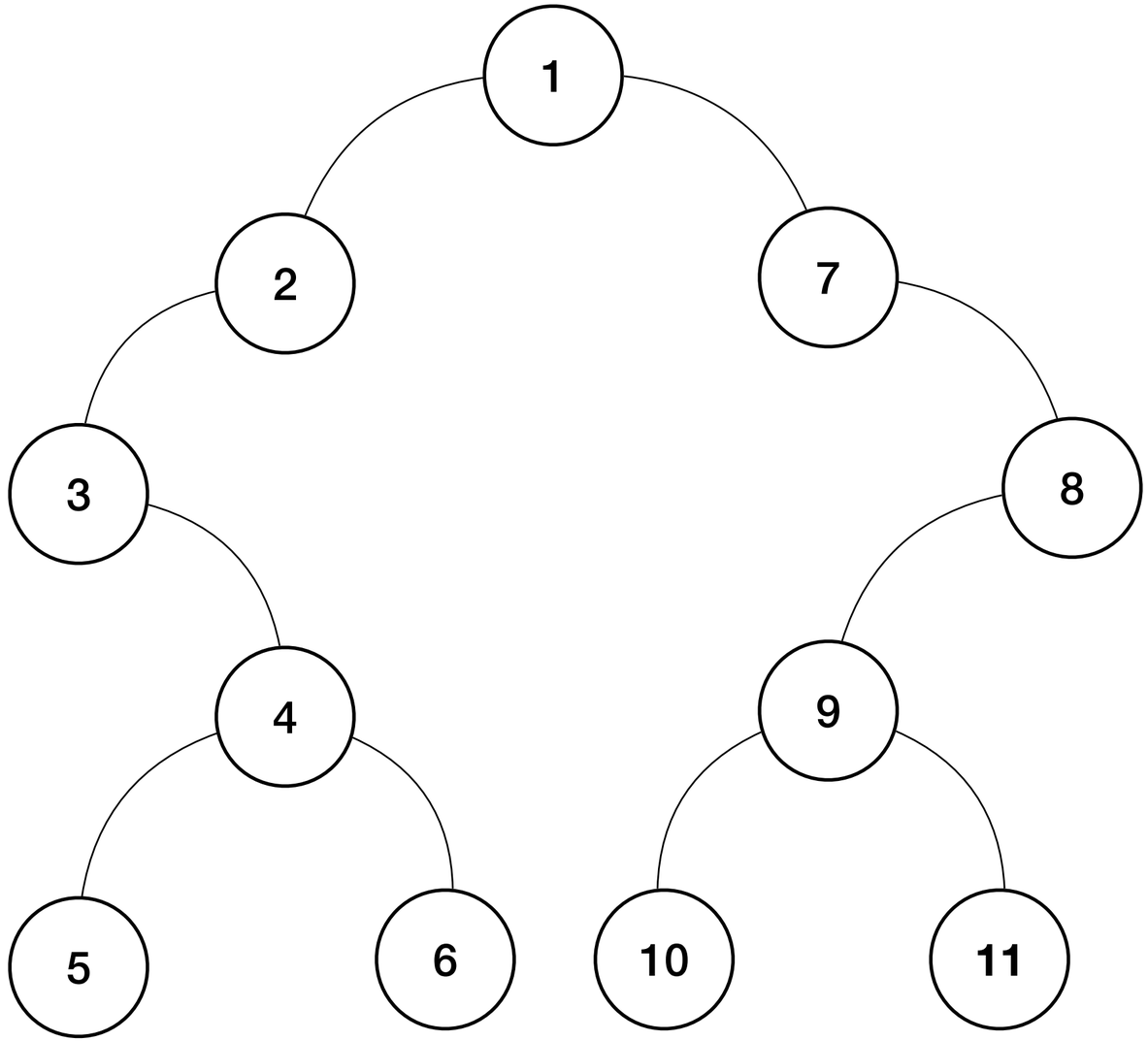


Review exercises - tree

- 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
  - Right boundary
- Should not repeat nodes

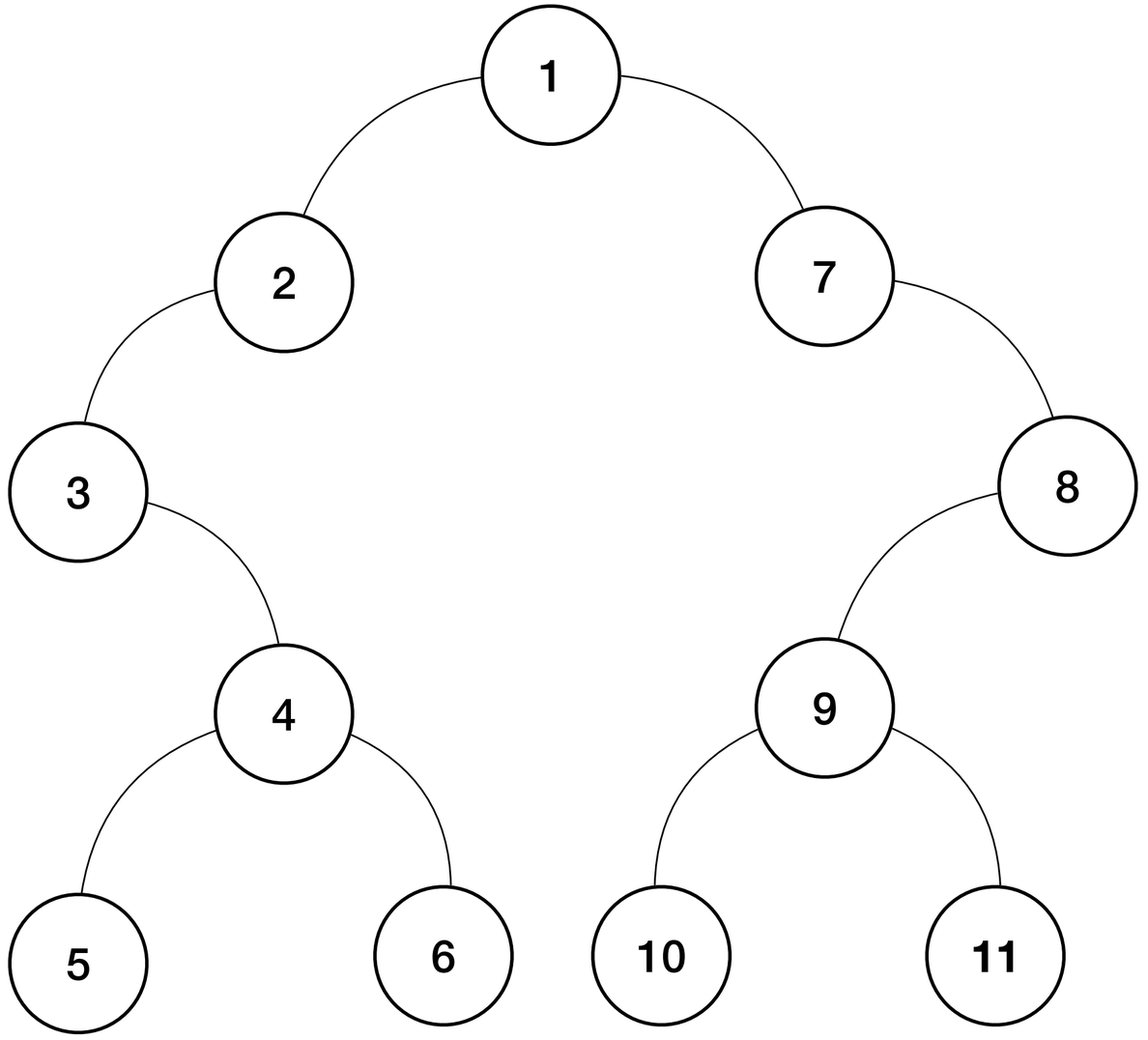


Review exercise - tree



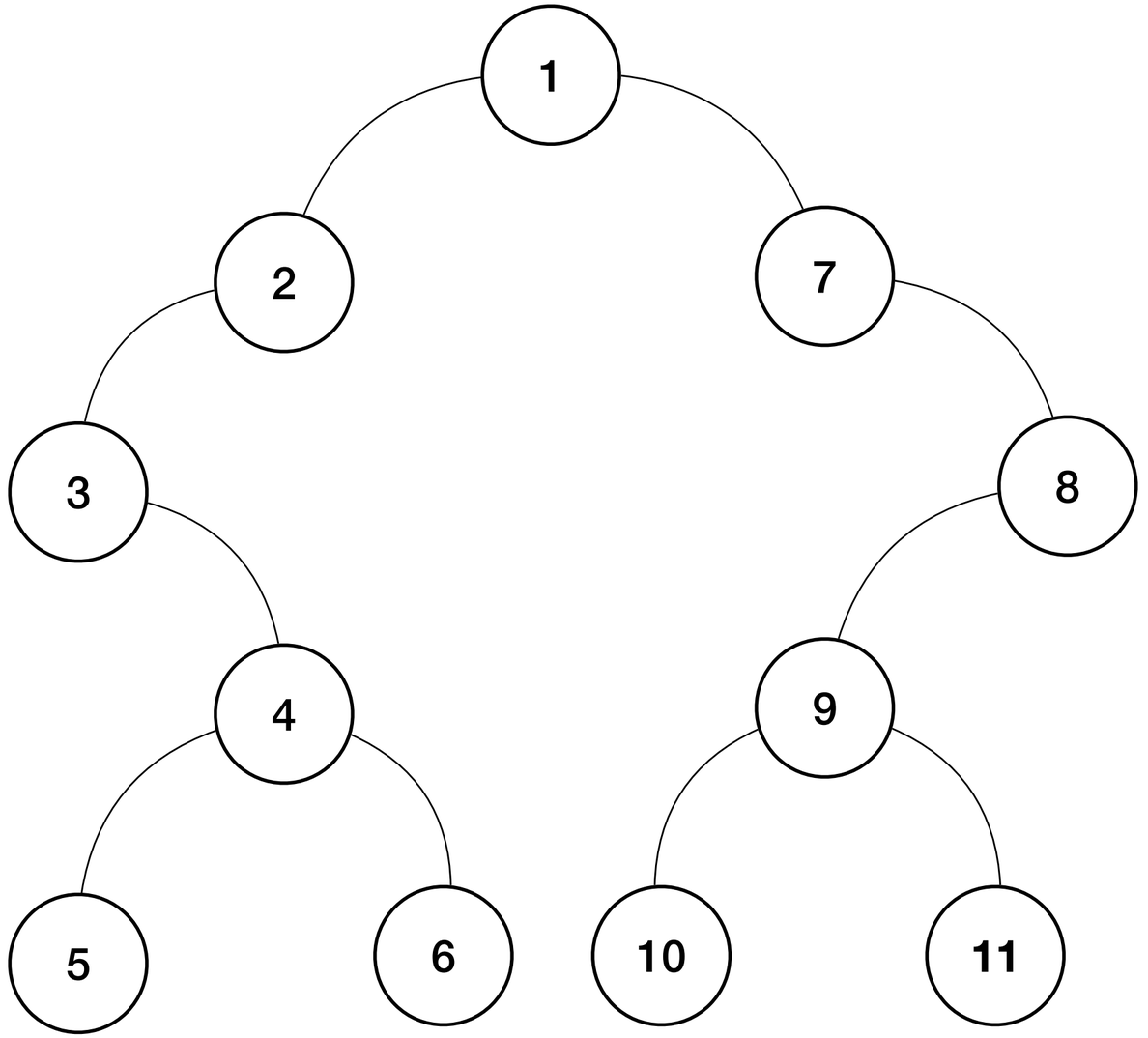
Review exercise - tree

- Delineate the left boundary



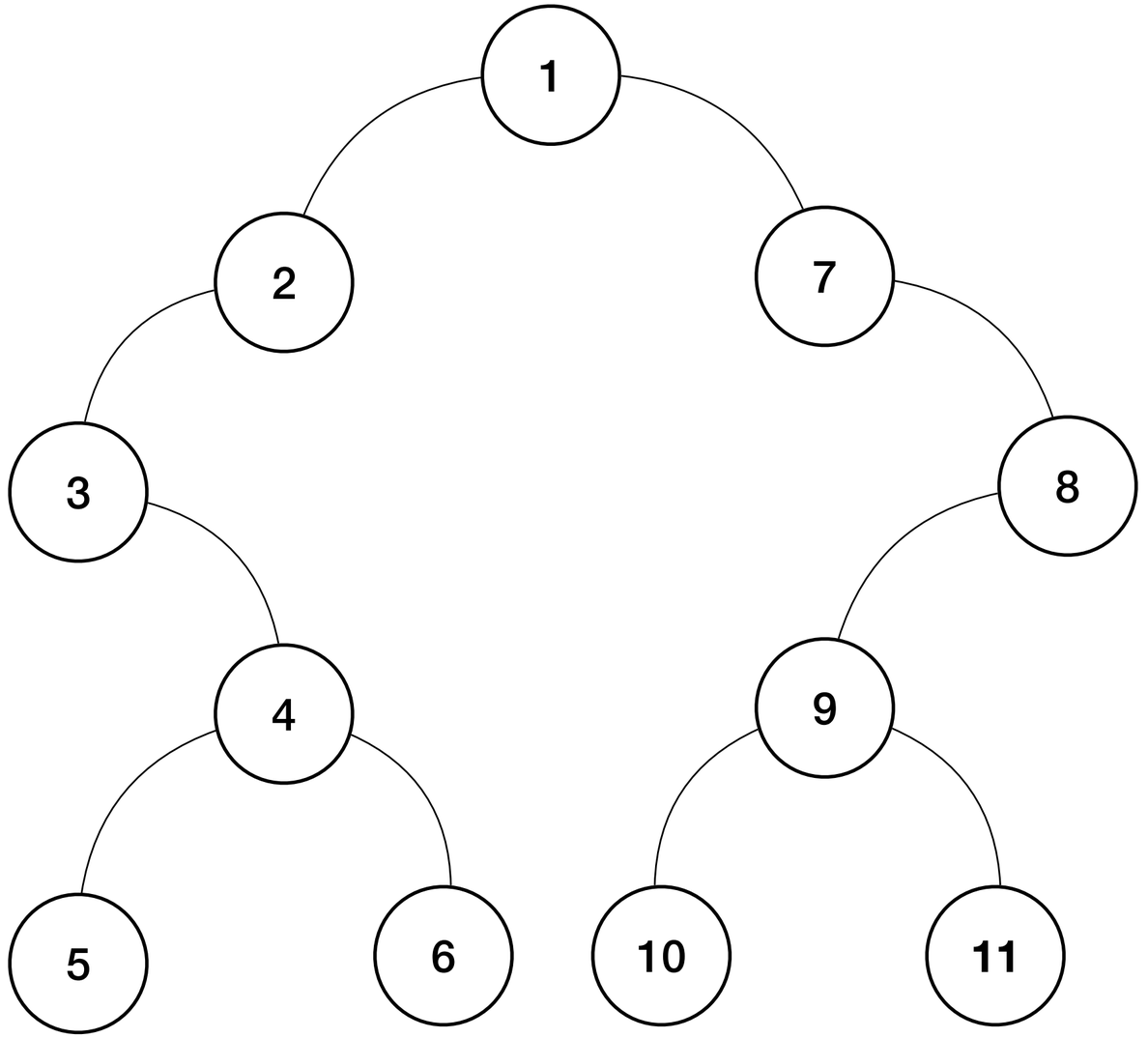
Review exercise - tree

- Delineate the left boundary
  - [2, 3, 4]



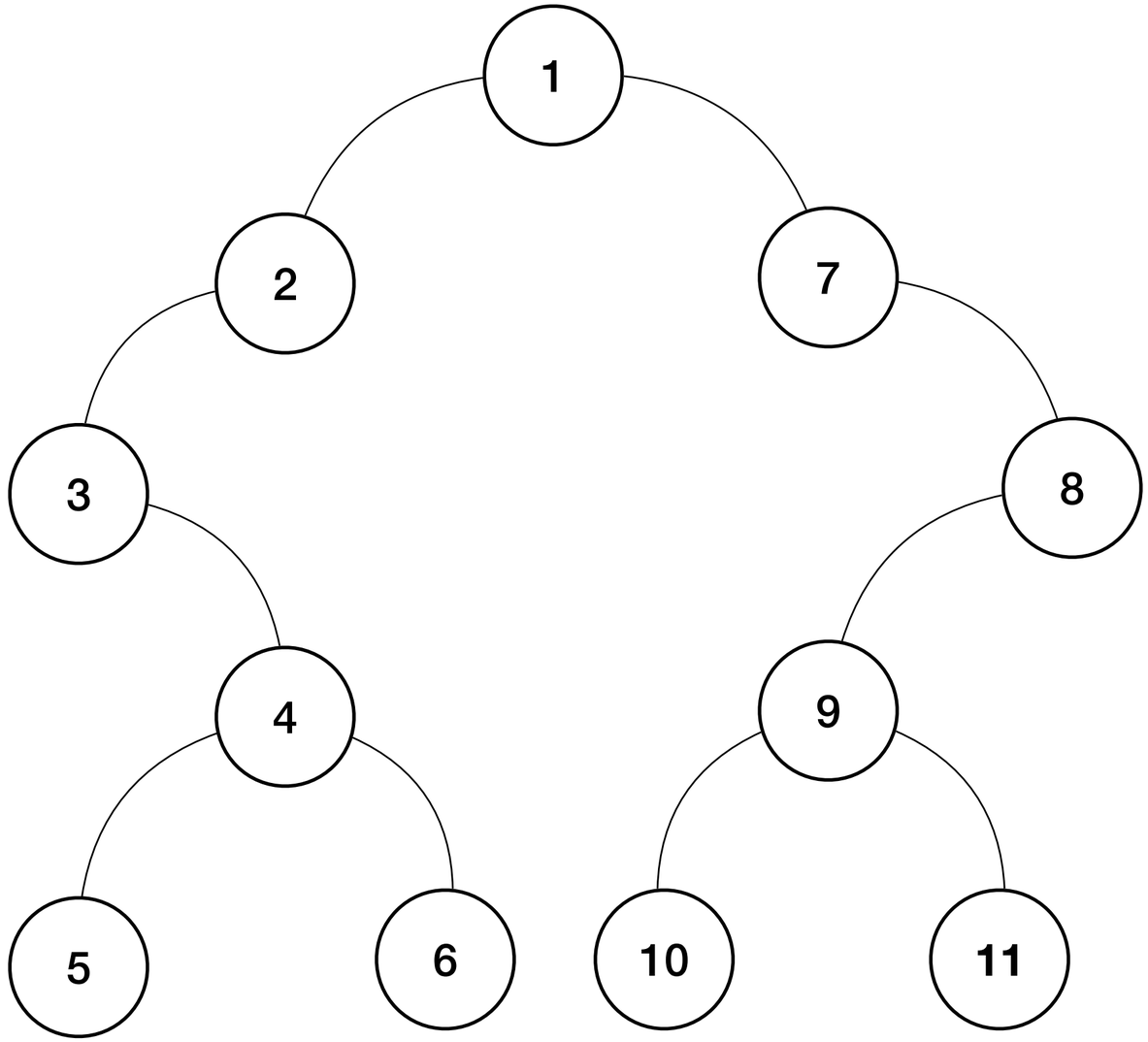
Review exercise - tree

- Delineate the left boundary
  - [2, 3, 4]
- Delineate the leaves



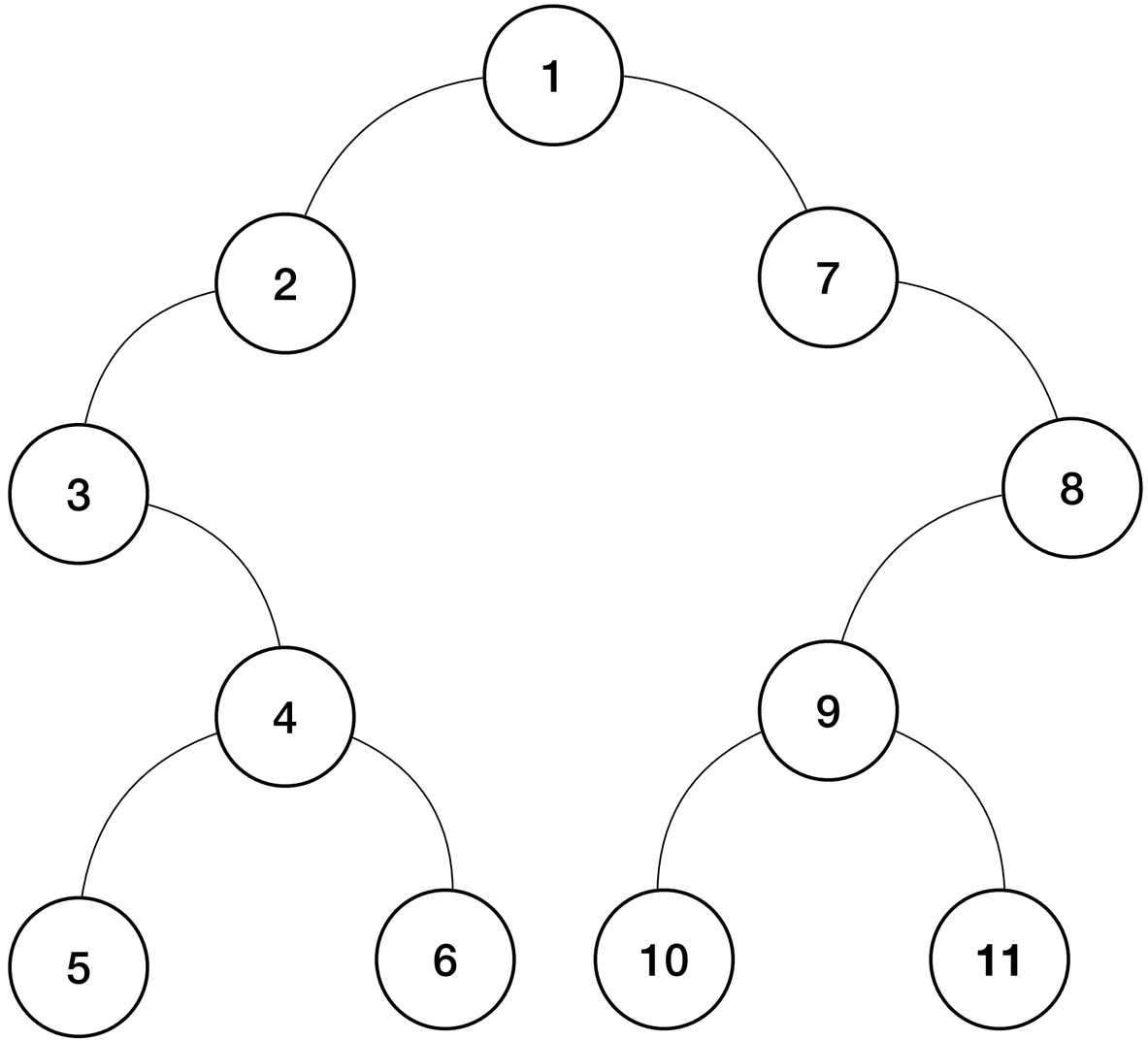
Review exercise - tree

- Delineate the left boundary
  - [2, 3, 4]
- Delineate the leaves
  - [5, 6, 10, 11]



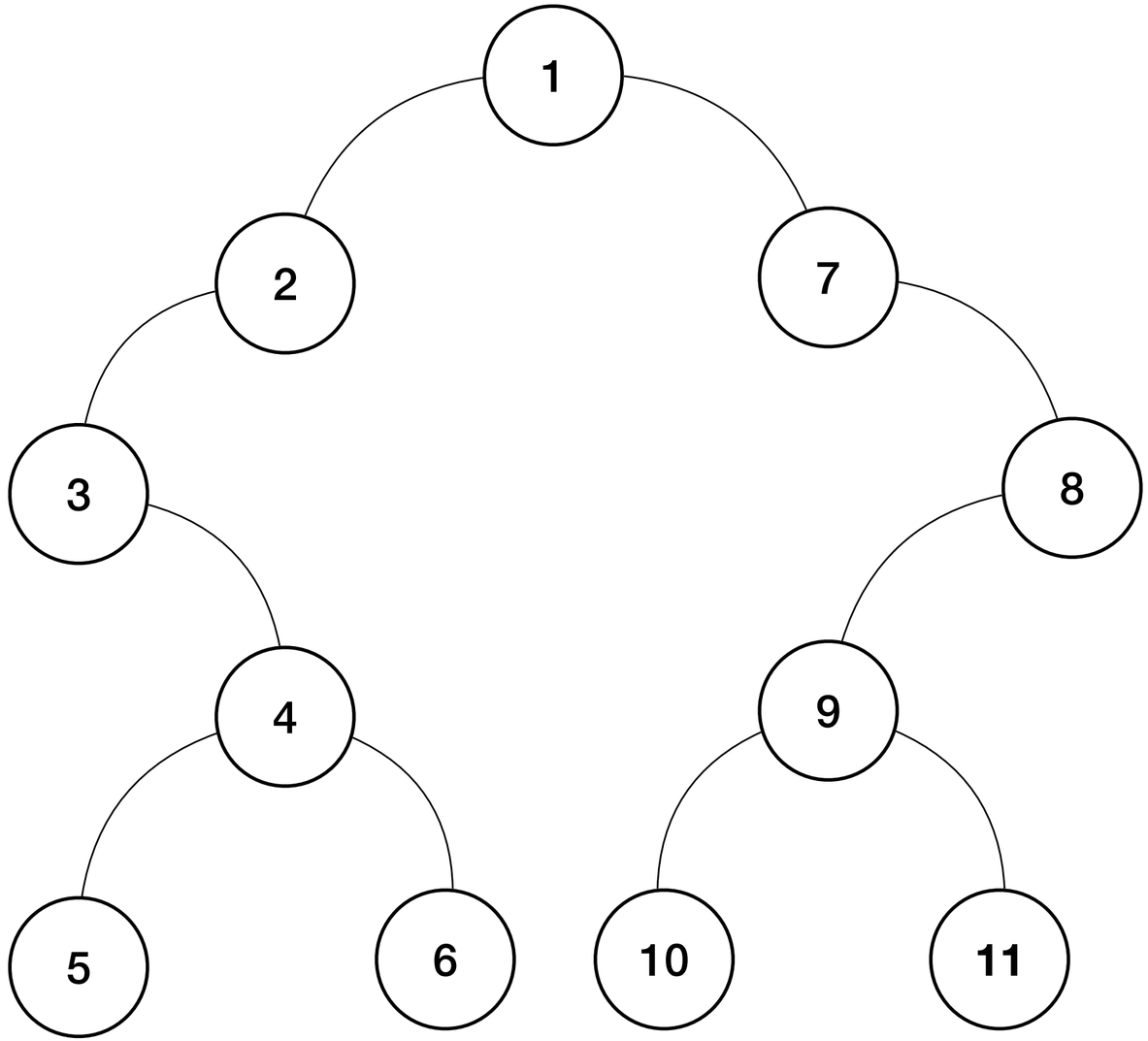
Review exercise - tree

- Delineate the left boundary
  - [2, 3, 4]
- Delineate the leaves
  - [5, 6, 10, 11]
- Delineate the right boundary



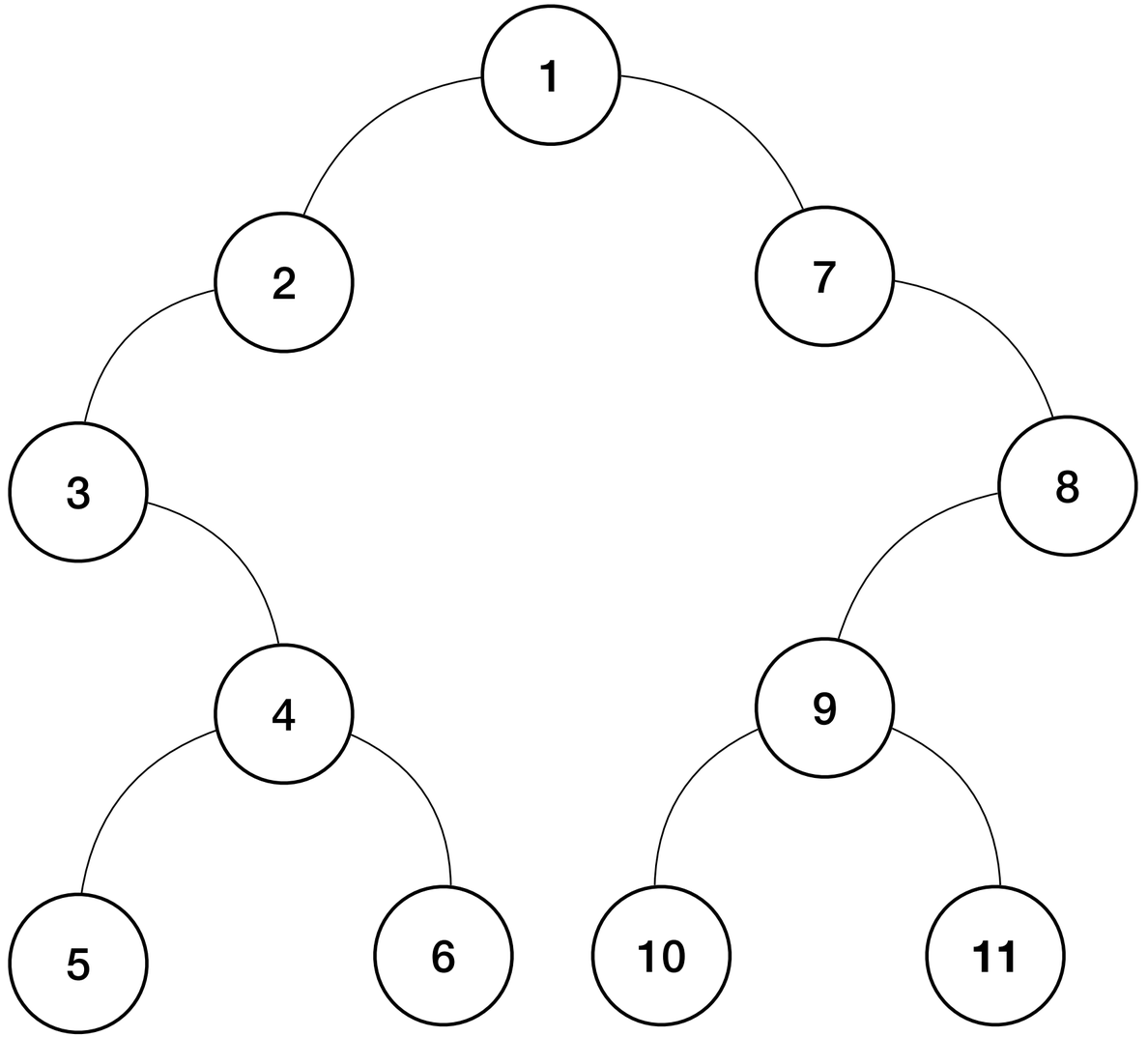
Review exercise - tree

- Delineate the left boundary
  - [2, 3, 4]
- Delineate the leaves
  - [5, 6, 10, 11]
- Delineate the right boundary
  - [9, 8, 7]



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



https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24/-/tree/main/lec0425

Review exercises

Review exercises

```
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 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_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 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_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 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);
}
```

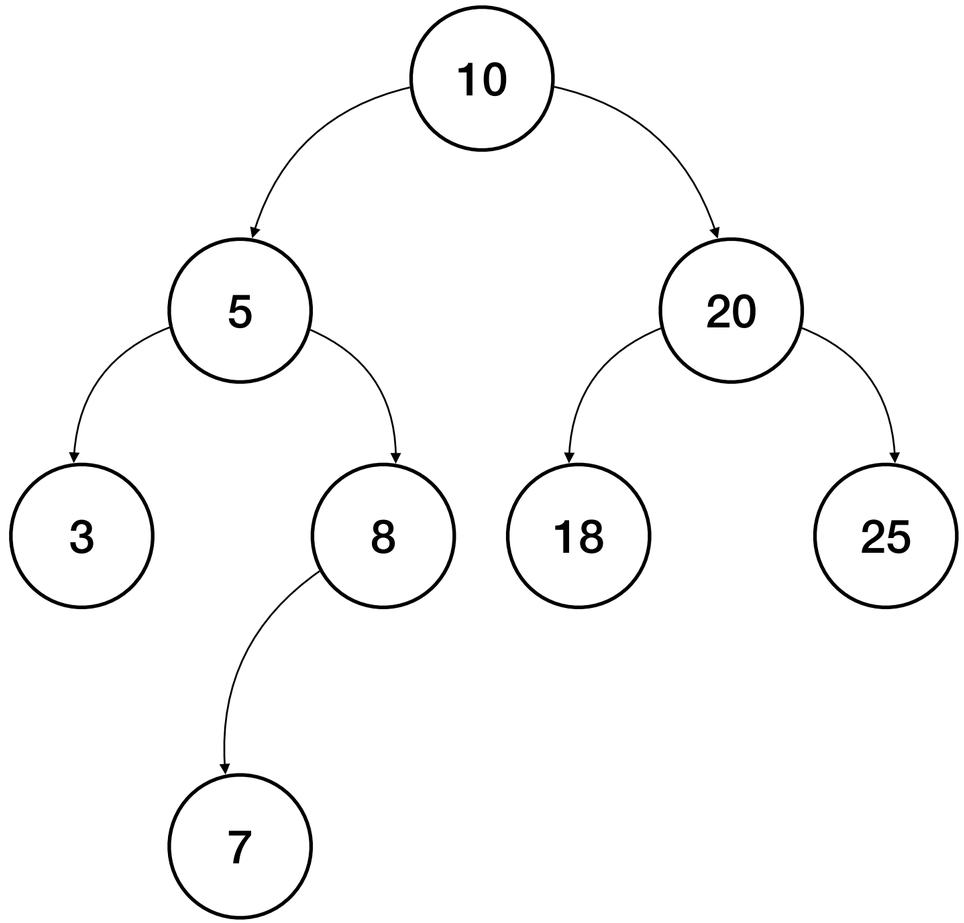
```
void print_leaves(node *cursor){
  if (cursor==NULL)
    return;
  print_leaves(cursor->left);
  if (cursor->left==NULL && cursor->right==NULL)
    cout<<cursor->data<<", ";
  print_leaves(cursor->right);
}
```

```
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 <<", ";
}
```

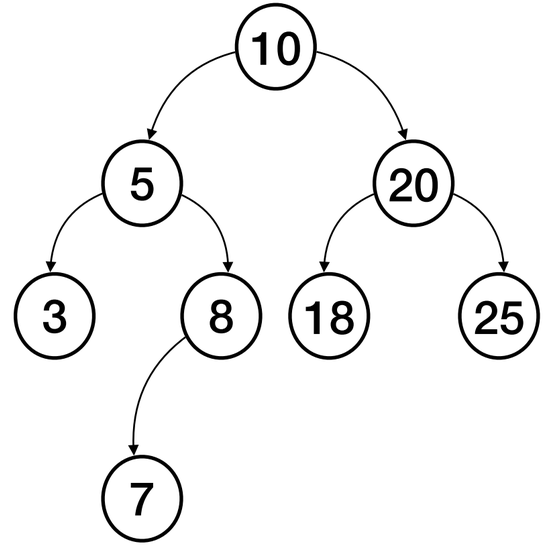
```
void print_boundary(node *cursor){
    if (cursor == NULL)
        return;
    cout << cursor->data << " ";

    // 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?

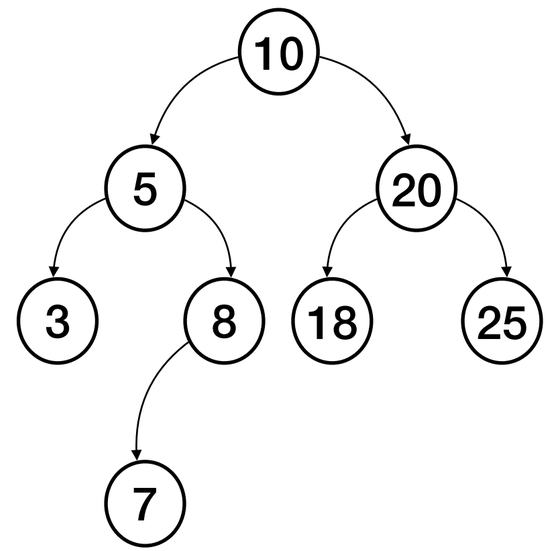


Review exercise - tree



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

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);

}
```

https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24/-/tree/main/lec0425

Review exercises - linked list

Review exercises - linked list

- Given a singly linked list, return a new *reduced* (remove all values matching some condition) linked list in *reverse* order.

Review exercises - linked list

- Given a singly linked list, return a new *reduced* (remove all values matching some condition) linked list in *reverse* order.

- Example:

Review exercises - linked list

- Given a singly linked list, return a new *reduced* (remove all values matching some condition) linked list in *reverse* order.

- Example:
  - Reduce and reverse

Review exercises - linked list

- Given a singly linked list, return a new *reduced* (remove all values matching some condition) linked list in *reverse* order.

- Example:
  - Reduce and reverse

1→2→5→3→2→5→7→8→9

Review exercises - linked list

- Given a singly linked list, return a new *reduced* (remove all values matching some condition) linked list in *reverse* order.

- Example:
  - Reduce and reverse

    1→2→5→3→2→5→7→8→9

    by removing even numbers.

Review exercises - linked list

- Given a singly linked list, return a new *reduced* (remove all values matching some condition) linked list in *reverse* order.

- Example:
  - Reduce and reverse

    1→2→5→3→2→5→7→8→9

    by removing even numbers.

- **Result**

Review exercises - linked list

- Given a singly linked list, return a new *reduced* (remove all values matching some condition) linked list in *reverse* order.

- Example:
  - Reduce and reverse

    1→2→5→3→2→5→7→8→9

    by removing even numbers.

- **Result**

    9→7→5→3→5→1

https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24/-/tree/main/lec0425

Review exercises - linked list

https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24/-/tree/main/lec0425

Review exercises - linked list

- Class activity

Review exercises - linked list

- Class activity
  - Just kidding … I just made up the question, let us try to work it out.

Review exercises - linked list

- Class activity
  - Just kidding … I just made up the question, let us try to work it out.

1→2→5→3→2→5→7→8→9

Review exercises - linked list

- Class activity
  - Just kidding … I just made up the question, let us try to work it out.

1→2→5→3→2→5→7→8→9

```
struct node{
  int data;
  struct node *next;
  node() : data(-1), next(NULL) {};
  node (int d) : data(d), next(NULL) {};
  node (int d, node *n) : data(d), next(n) {};
};
```

Review exercises - linked list

- Class activity
  - Just kidding … I just made up the question, let us try to work it out.

1→2→5→3→2→5→7→8→9

```
struct node{
  int data;
  struct node *next;
  node() : data(-1), next(NULL) {};
  node (int d) : data(d), next(NULL) {};
  node (int d, node *n) : data(d), next(n) {};
};
```

```
node * reduce_list(node *cursor){
  node *new_list = NULL;
  while (cursor){
    if (cursor->data%2==1)
      new_list = new node(cursor->data, new_list);
    cursor=cursor->next;
  }
  return new_list;
}
```