

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  
1C3015C0 01010100 30011100 00002020 20202E4F 52494720 20207833 3030300A E0001300 00002020 20204C45 41202052  
302C206D 794C696E 6509E200 13000000 20202020 4C454120 2052312C 206D794C 696E6540 60001600 00004C4F 4F502020  
20204C44 52205230 2C205231 2C202330 21F00010 00000020 20202020 20202054 52415020 78323105 24001400 00002020  
20202020 20204C44 20205232 2C207465 726D8014 00160000 00202020 20202020 20414444 2052322C 2052322C 20523002  
04001000 00002020 20202020 20204252 7A203B54 F50612 00150000 02202020 20202020 20414444 2052312C 2052312C  
2031F90F 00120000 00202020 20202020 20425252 7A203B54 F50612 00150000 02202020 00005354 4F502020 20204841 4C54D0FF  
00150000 00746572 6D202020 202E4649 4C4C2020 20784646 44306900 00010000 00697400 00010000 00746100 00010000  
00616200 00010000 00627200 00010000 00726100 00010000 00015800 00010000 00683200 00010000 00324000 00010000  
00406600 00010000 00666100 00010000 00520000 00010000 00010000 00332D00 00010000 002D6500 00010000  
00656300 00010000 00636500 00010000 00653200 00010000 00323200 00010000 00323000 00010000 00300000 002A0000  
006D794C 696E6520 202E5354 52494E47 5A202020 20226974 61627261 68324066 6132332D 65636532 32302200 00000000  
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

# 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  $\sim$  TRAP vector
  - RTI used to return

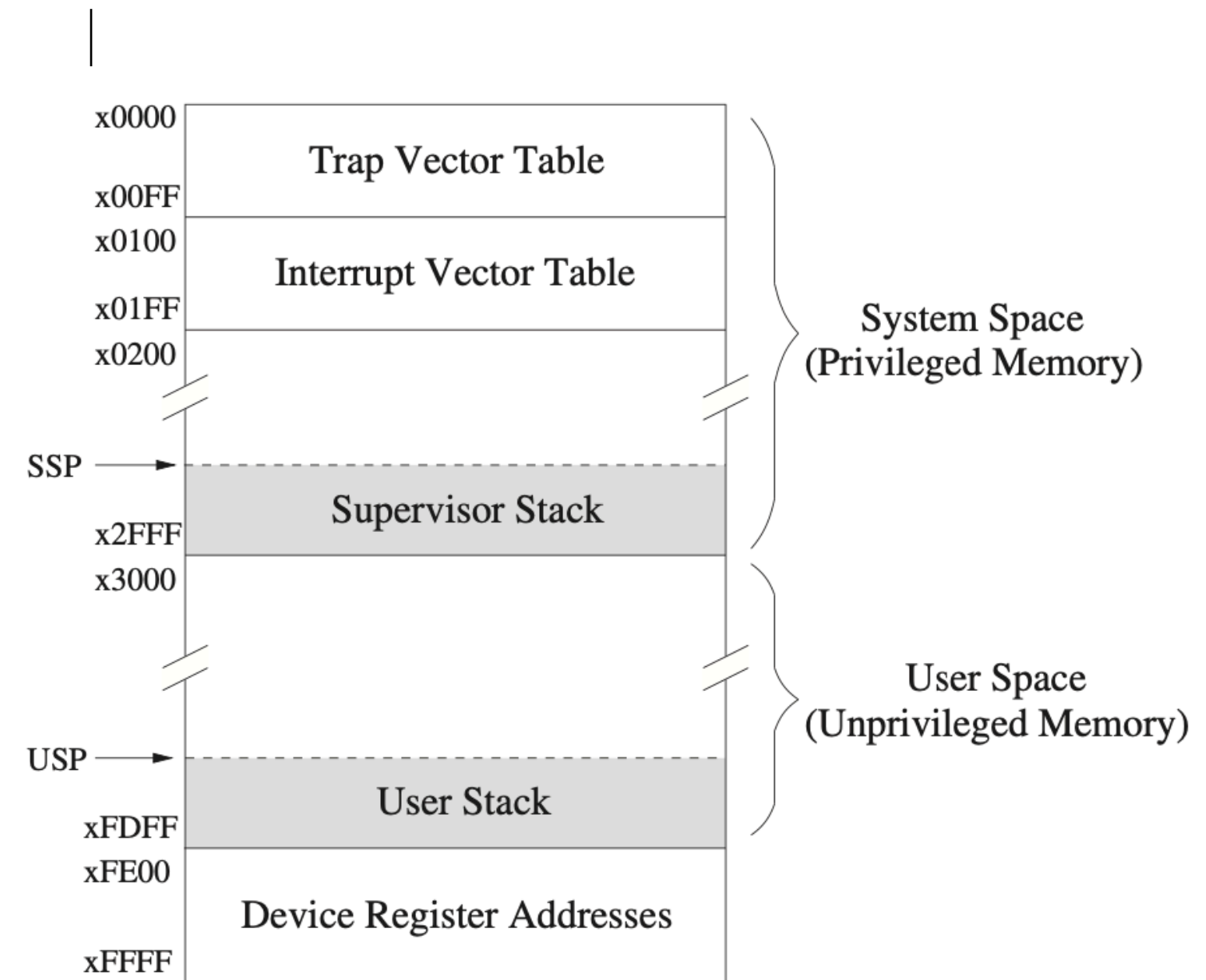
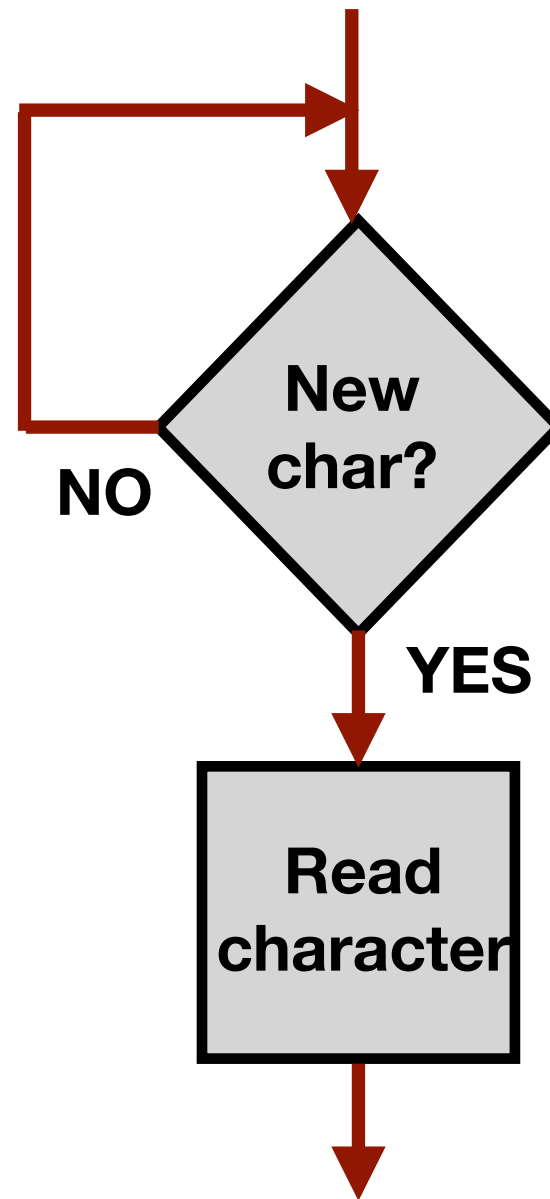
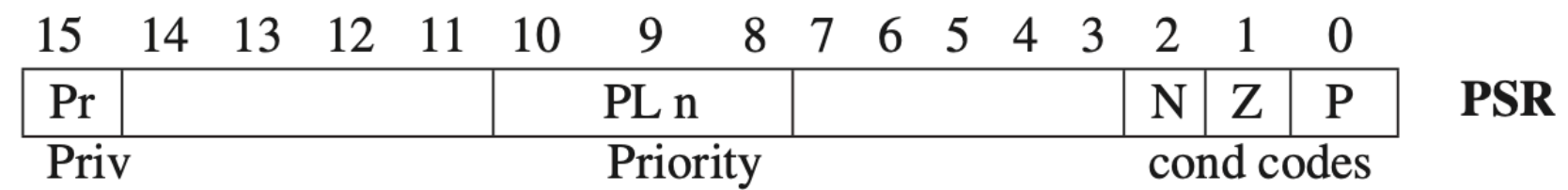


Figure A.1 - P&P 3rd Ed.



# Recap - Interrupts

- What needs to be saved?
  - PC, PSR
  - R6  $\leftrightarrow$  Saved\_USP, Saved\_SSP

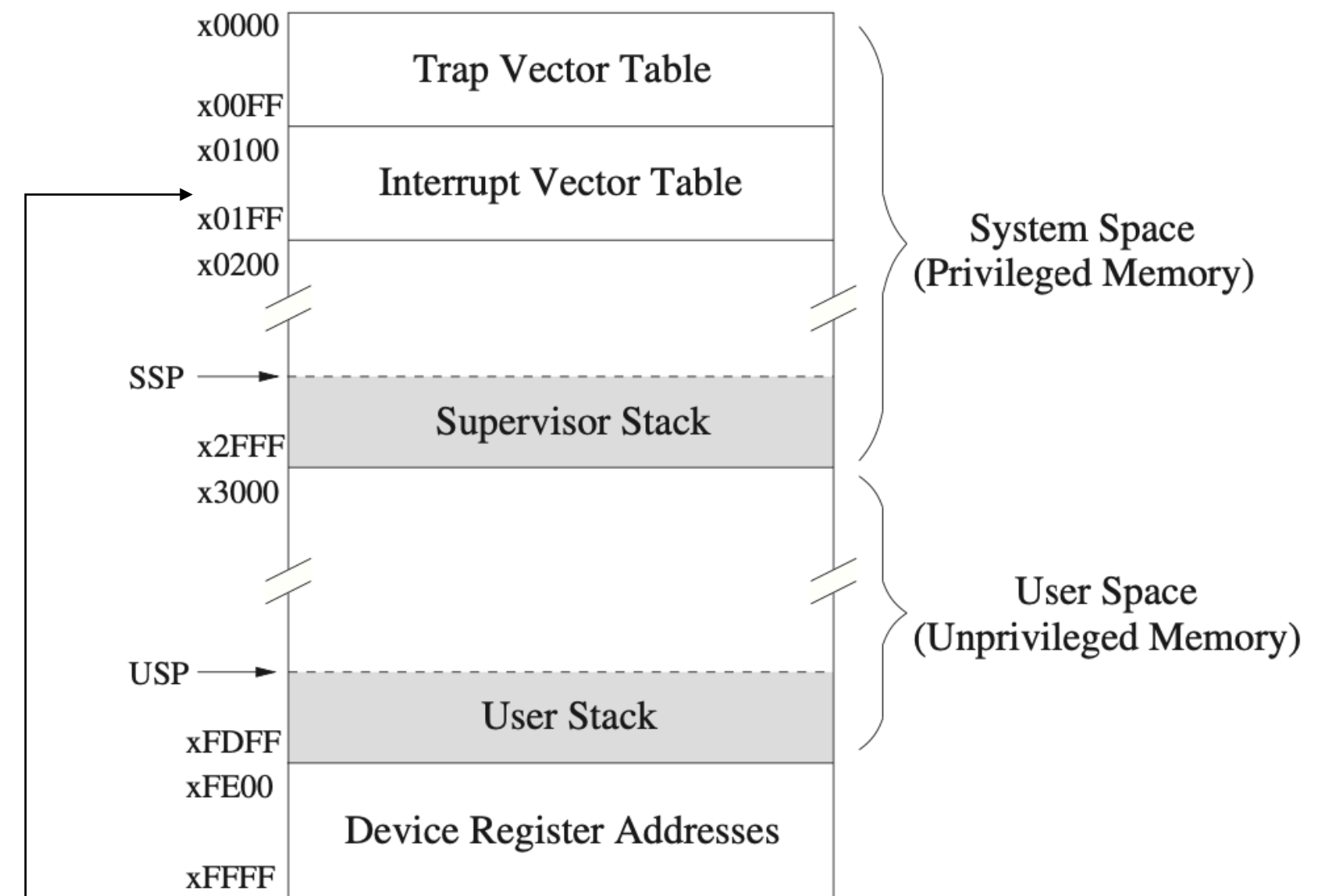
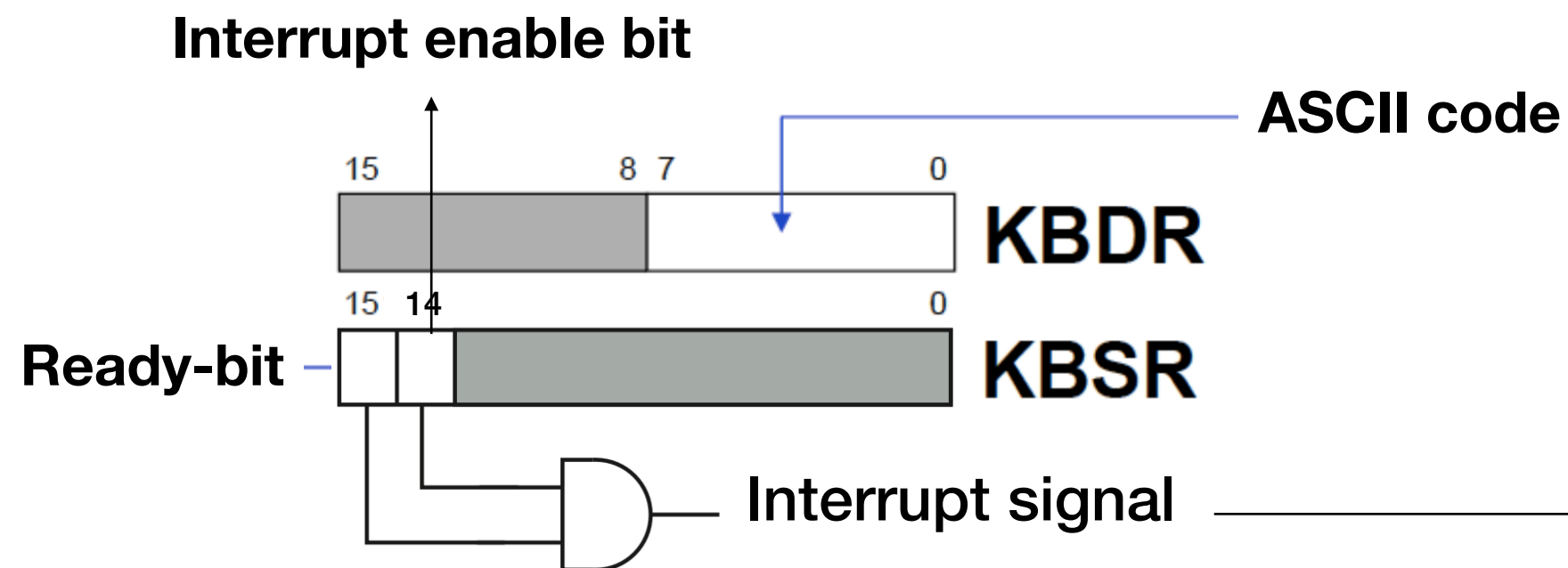


Figure A.1 - P&P 3rd Ed.

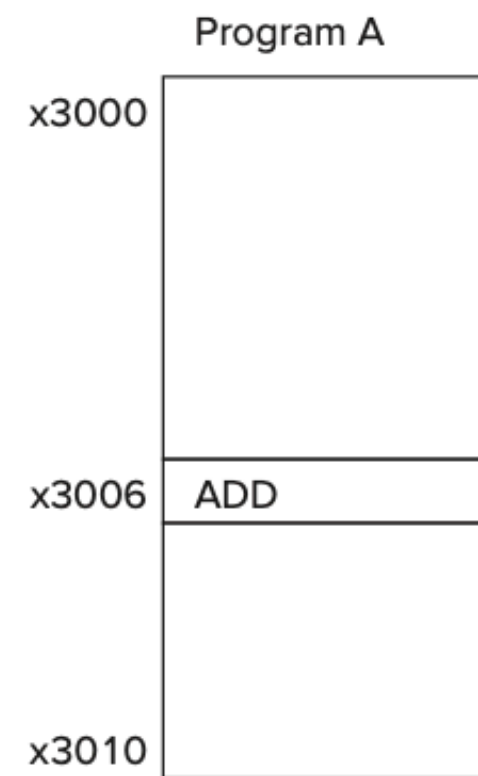
# Interrupt example

```
.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

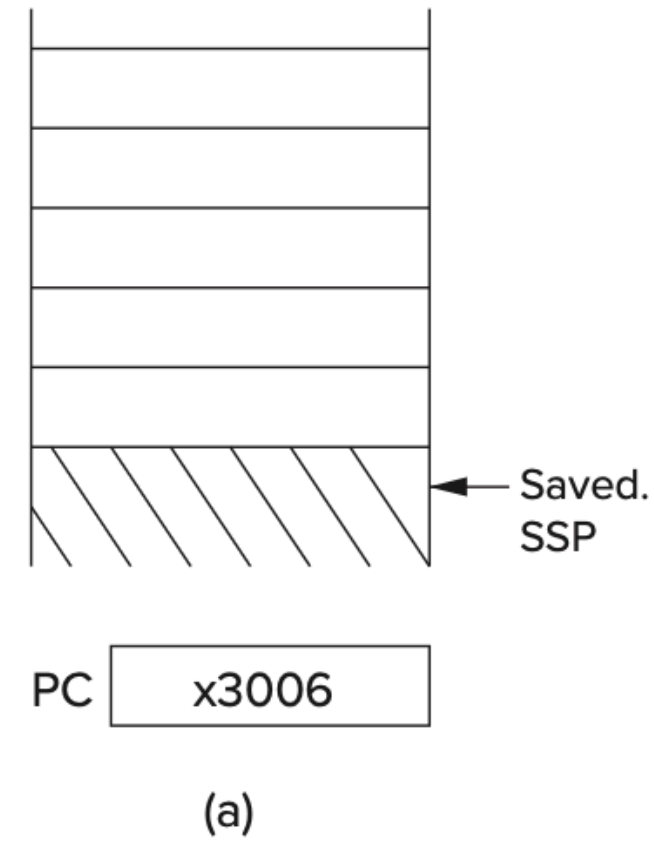
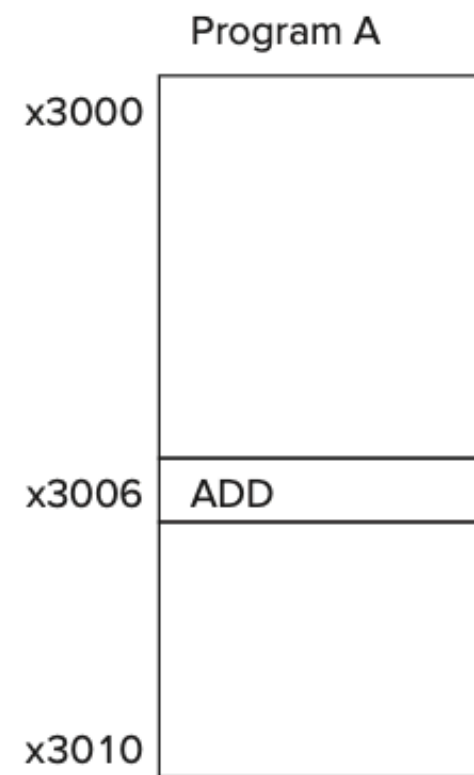
;
EN_IE   .FILL  x4000    ; To enable the IE bit
NUM2    .FILL  x0032    ; ASCII Code for '2'
KBSR    .FILL  xFE00
KBDR    .FILL  xFE02
KBINTV  .FILL  x0180    ; INT vector table address for keyboard
SaveR0  .BLKW  #1
.END
```

What does this program do?

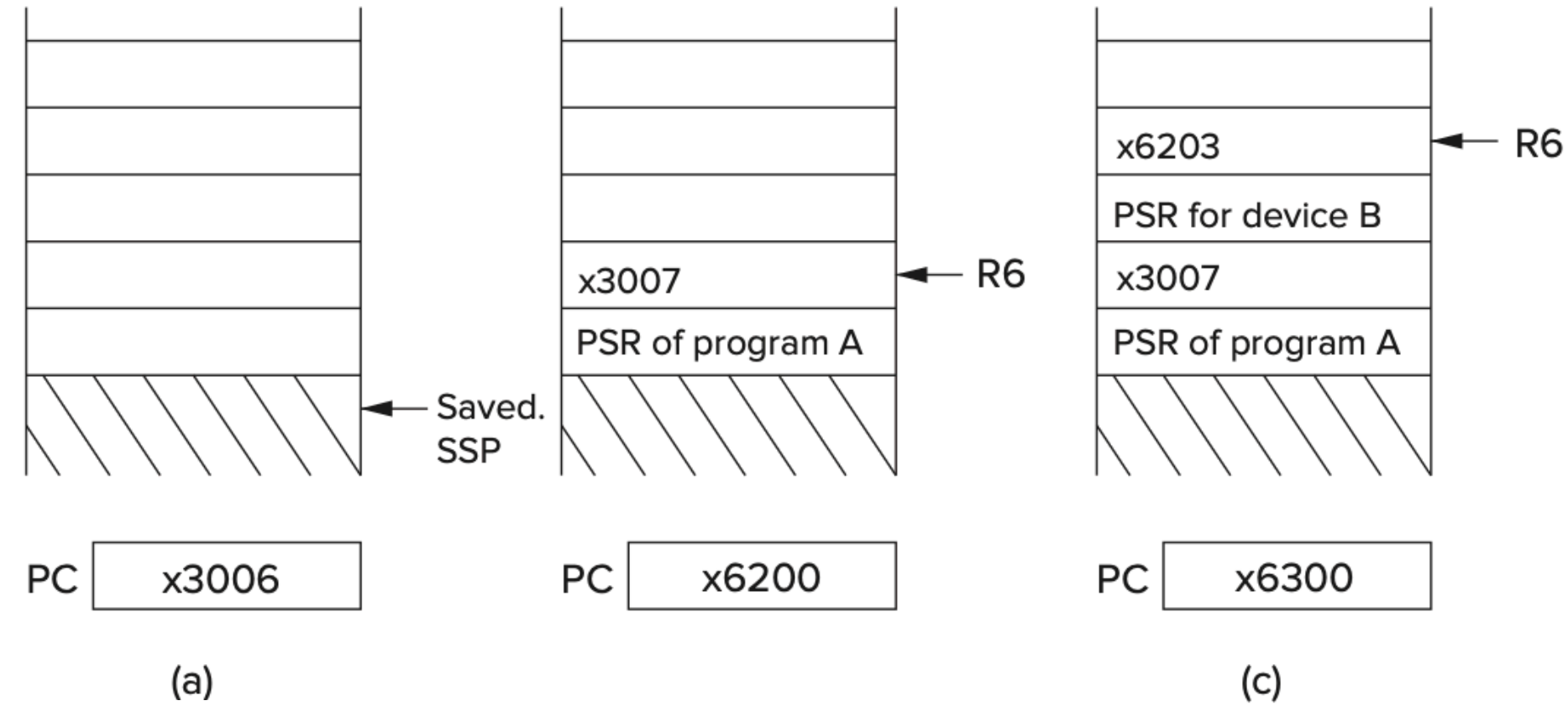
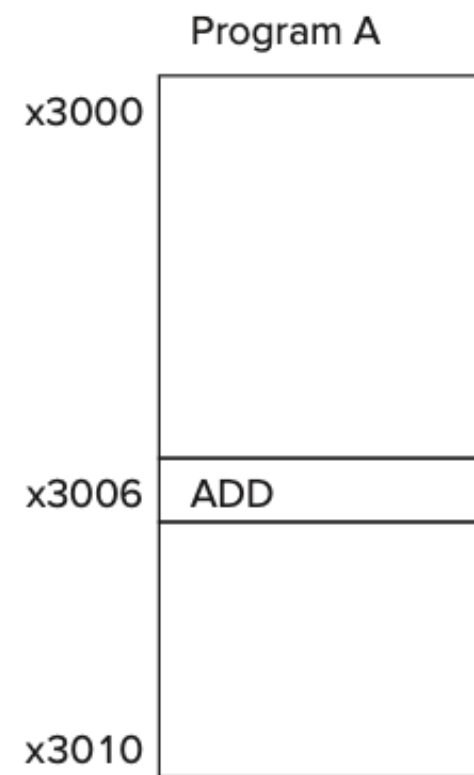
# Interrupting interrupts ...



# Interrupting interrupts ...



# 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.

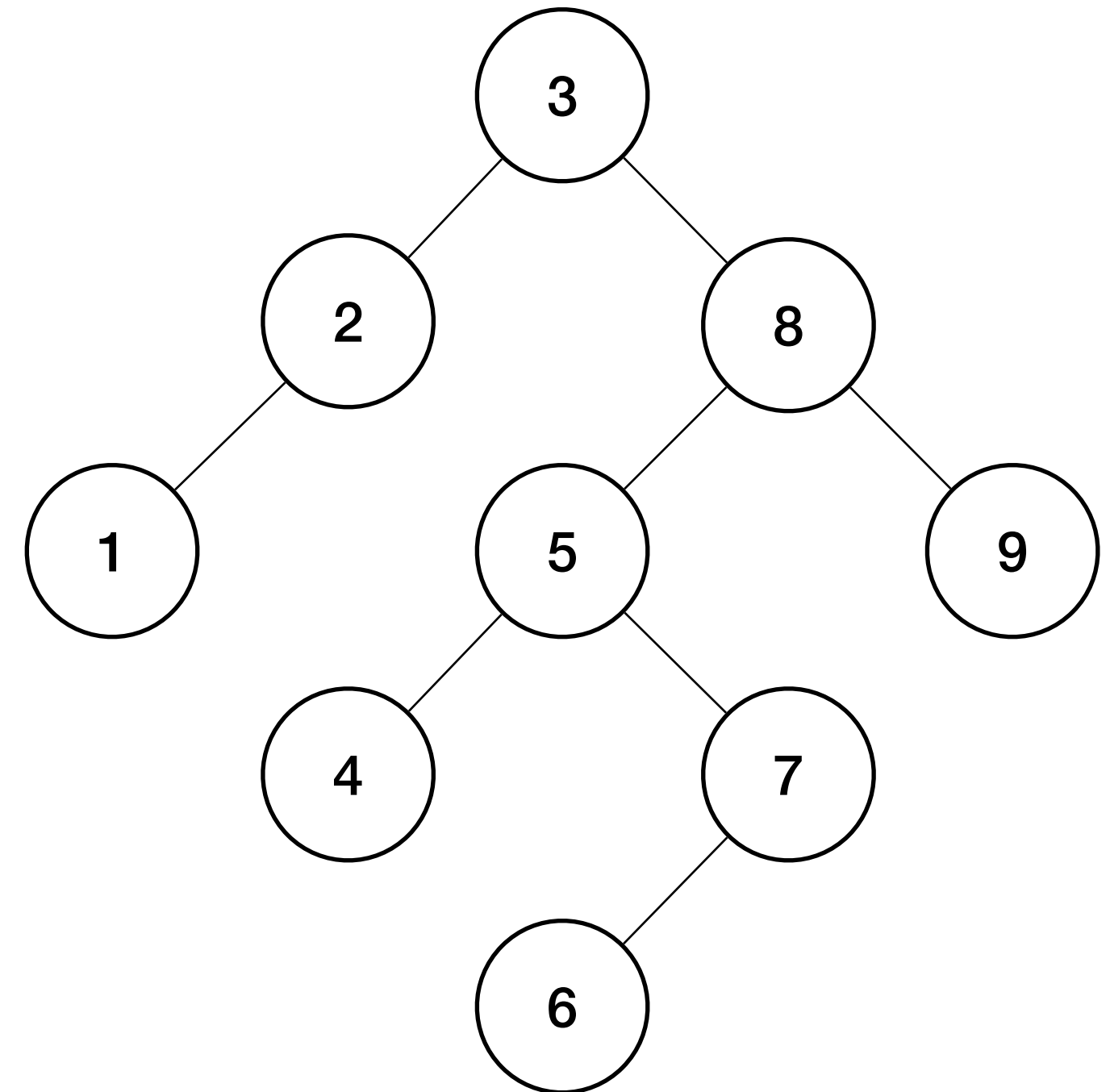
```
struct node{
    int data;
    struct node *left;
    struct node *right;
    node() : data(0), left(NULL), right(NULL) {};           Initializer list syntax
    node(int d) : data(d), left(NULL), right(NULL) {};
    node(int d, node *l, node *r) : data(d), left(l), right(r) {};
};
```

Overloaded  
constructors

```
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;

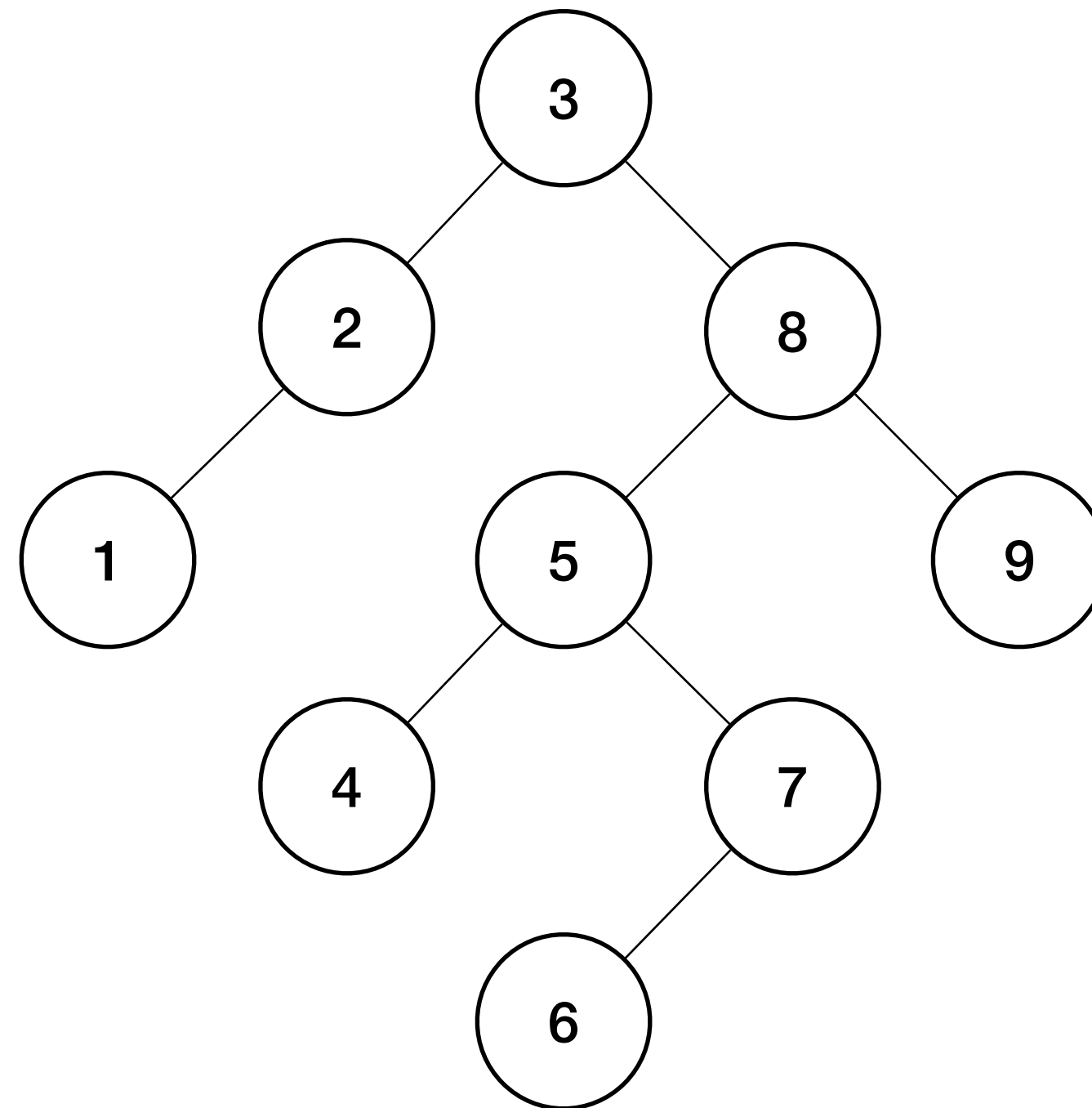
    left = new node(6);
    right = new node(7, left, NULL);
    left = new node(4);
    left = new node(5, left, right);

    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;

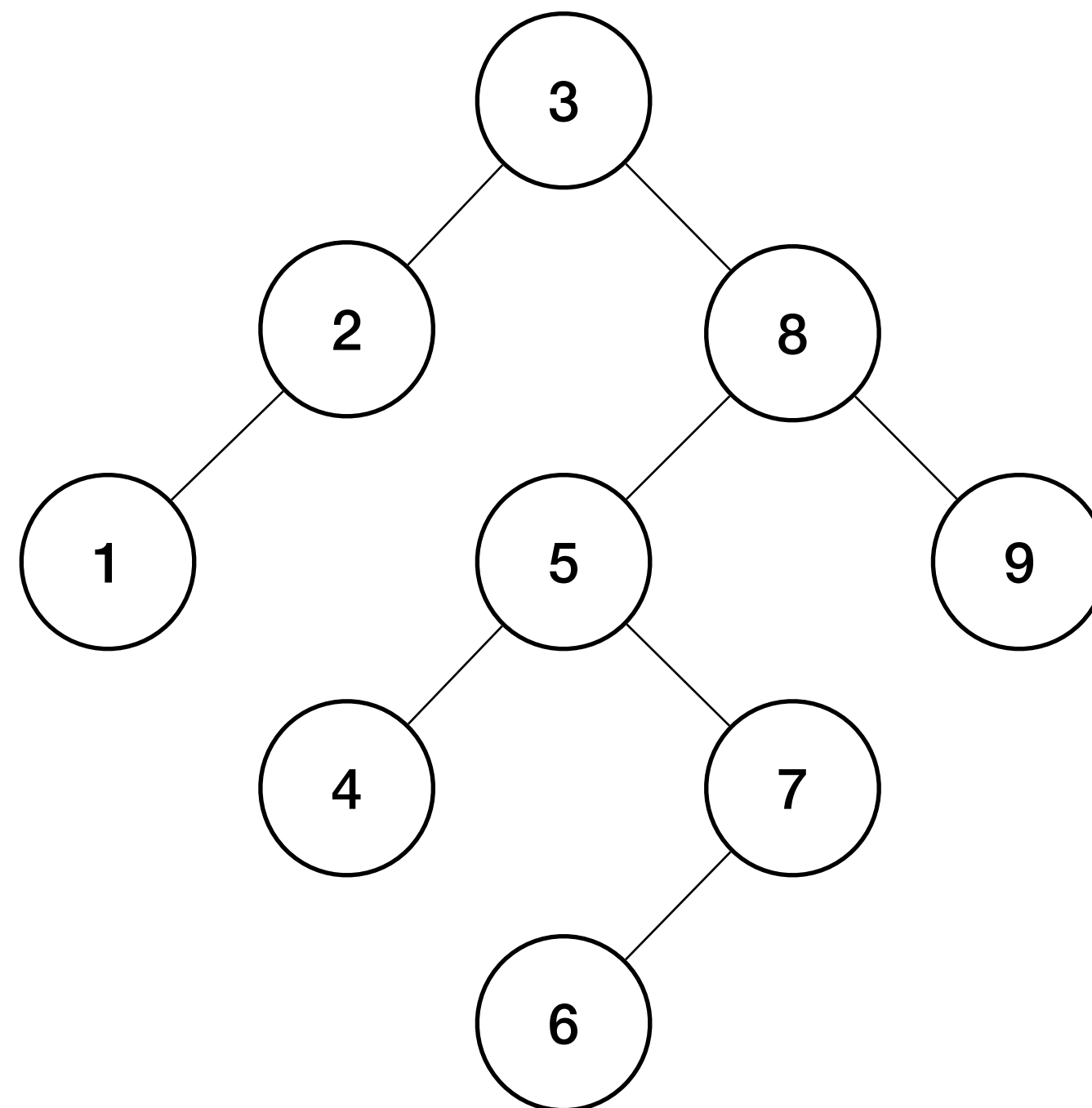
    left = new node(6);
    right = new node(7, left, NULL);
    left = new node(4);
    left = new node(5, left, right);

    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);
}

```

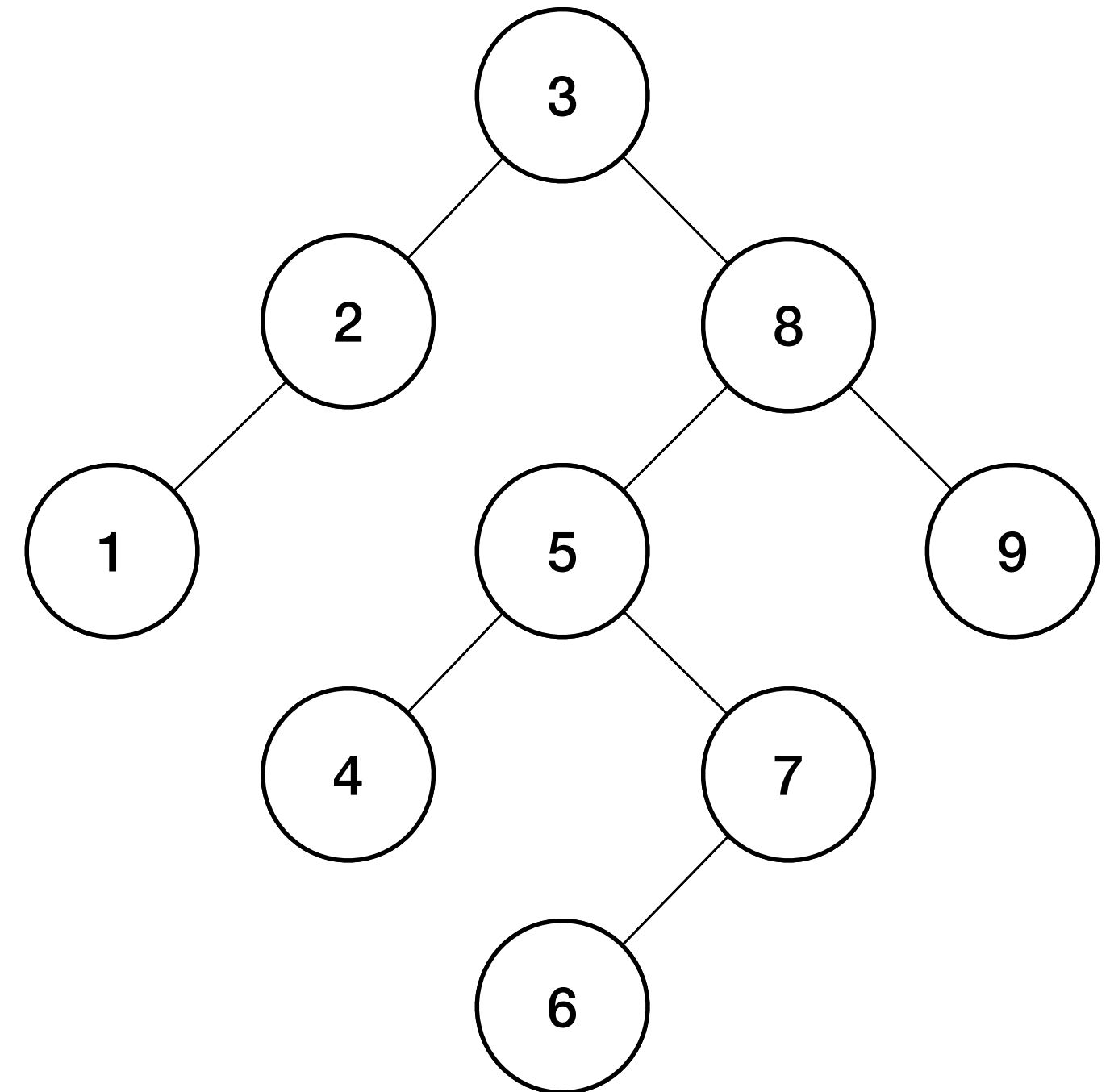


# Review exercise: linked list

- Implement a **circularly** linked ***sorted*** list:
  - Print list function
  - Add to list function
    - Add to empty list
    - Add in middle
      - Add as first
      - Add as last
  - Destroy list function

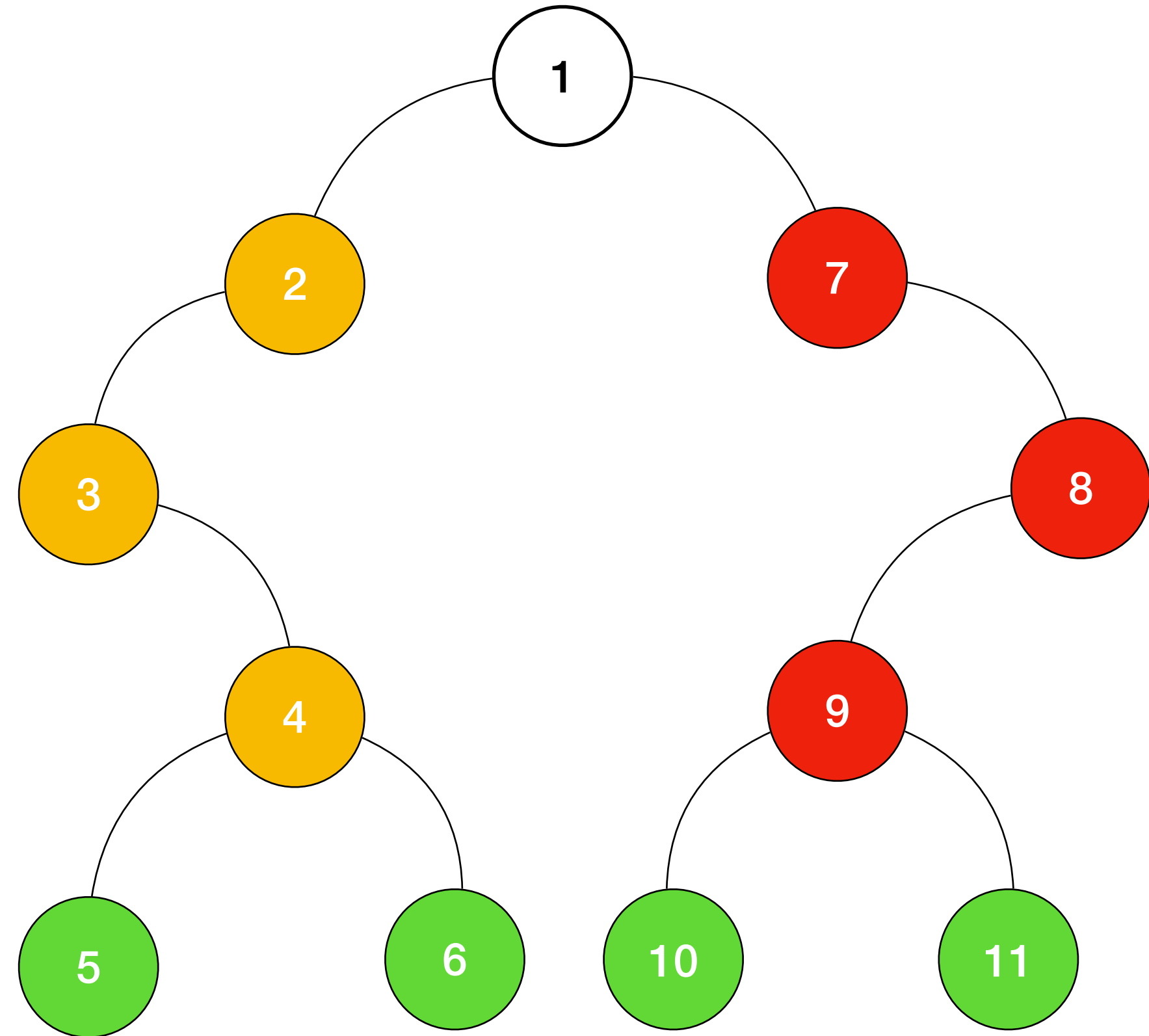
# Review exercises - tree

- 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 exists**
- 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 << " ";
```

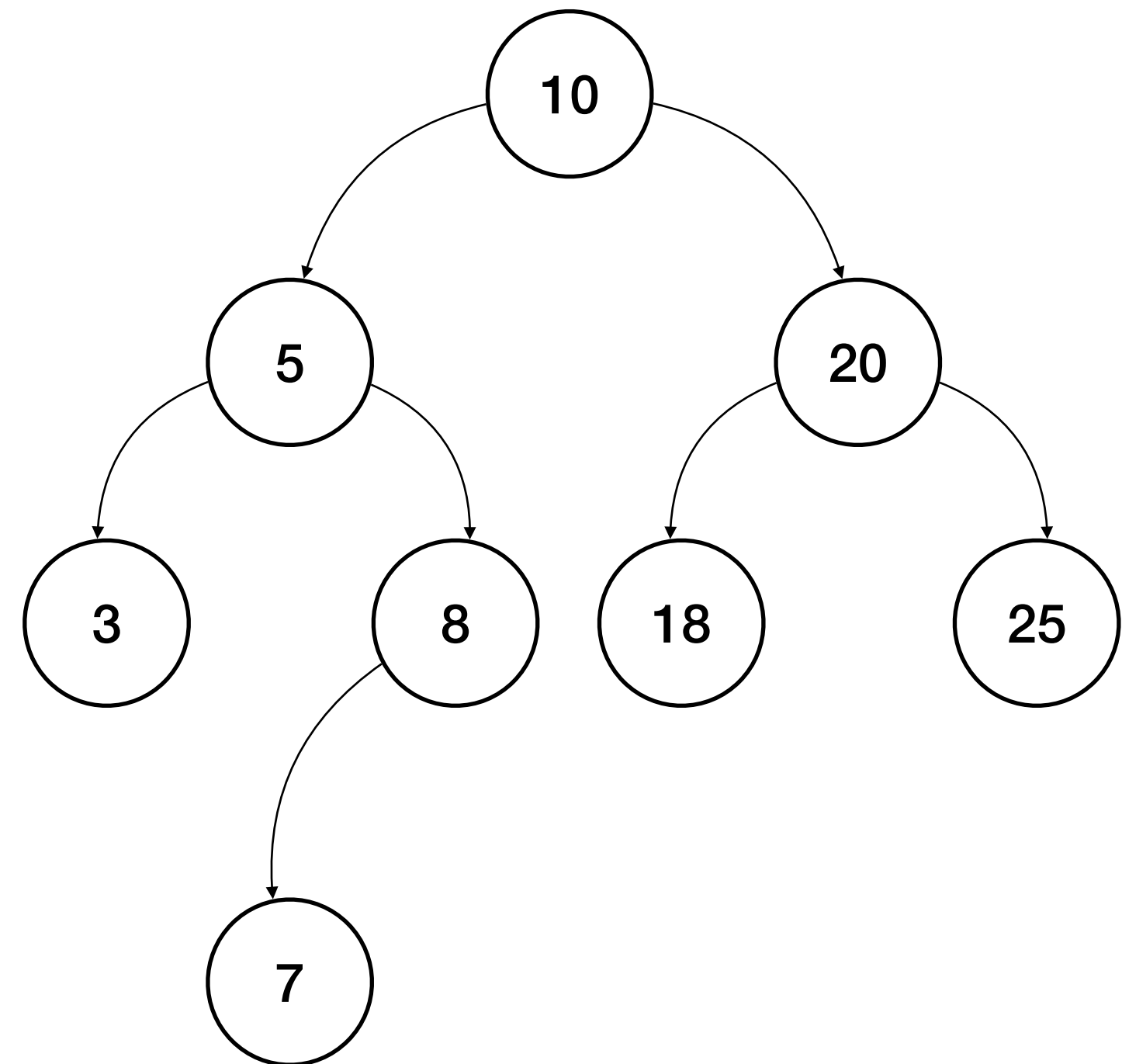
```
// Print the left boundary
print_left_boundary(cursor->left);
```

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

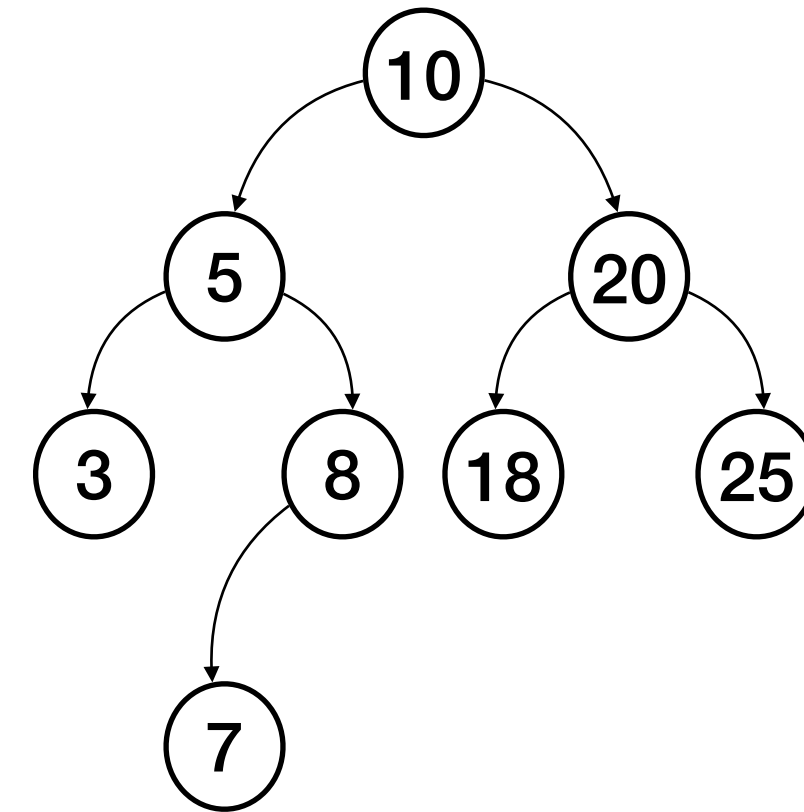
```
// Print the right boundary
print_right_boundary(cursor->right);
}
```

# 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);
}
```