CSE102
Computer Programming with C

2016-2017 Fall Semester

Dynamic Data Structures

© 2015-2016 Shahid Alam

Largely adapted from The C programming Language K&R and

Refresher – Pointers

- A pointer is a variable that contains the address of a variable.

- The two unary operators **&** and *. The **&** gives the *address* of the object and * *accesses* the object the pointer points to and is also called the dereferencing operator.

Pointers

```
int x = 1, y = 2, z[10];

int *ip;      /* ip is a pointer to int */
ip = &x;      /* ip now points to x */
y = *ip;
*ip = 0;
ip = &z[0]    /* ip now points to z[0] */
```

Pointers

```
int x = 1, y = 2, z[10];

int *ip;        /* ip is a pointer to int */
ip = &x;        /* ip now points to x */
y = *ip;        /* y is now 1 */
*ip = 0;
ip = &z[0]      /* ip now points to z[0] */
```

Pointers

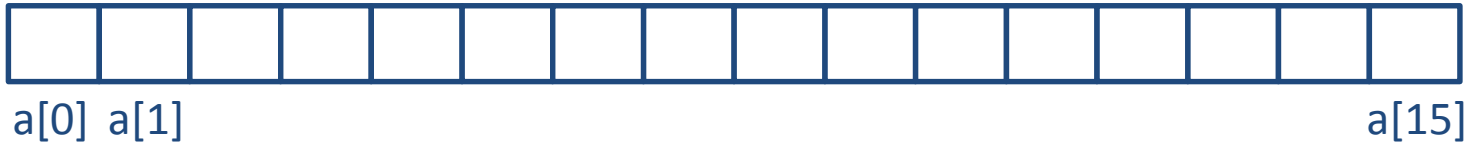
```
int x = 1, y = 2, z[10];

int *ip;        /* ip is a pointer to int */
ip = &x;        /* ip now points to x */
y = *ip;        /* y is now 1 */
*ip = 0;        /* x is now 0 */
ip = &z[0]      /* ip now points to z[0] */
```

Pointers

```
*ip = *ip + 10;  /* increments *ip by 10 */

y = *ip + 1;     /* add 1 to what ip points at,
                    and assigns the result to y */

*ip += 1;        /* increments what ip points to */
++*ip;
(*ip)++;

int *iq;
iq = ip;         /* copies the content of ip into
                    iq, thus making iq points to
                    whatever ip pointed to */
```

Pointers and Arrays

int a[16];



Pointers and Arrays

int a[16];

a[0] a[1] a[15]

int *pa;

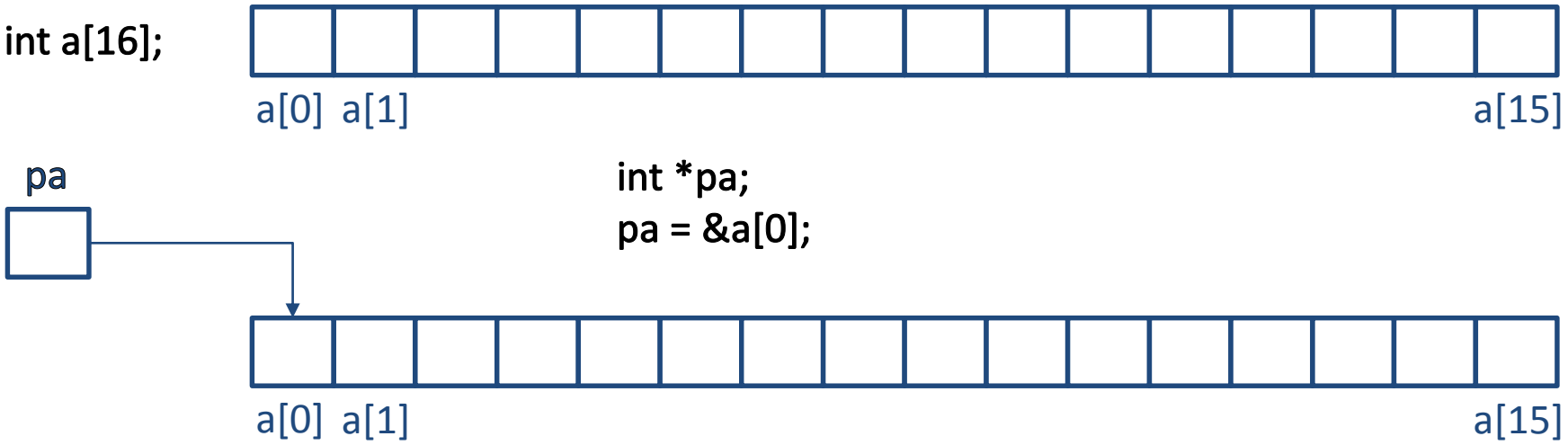
Pointers and Arrays

int a[16];

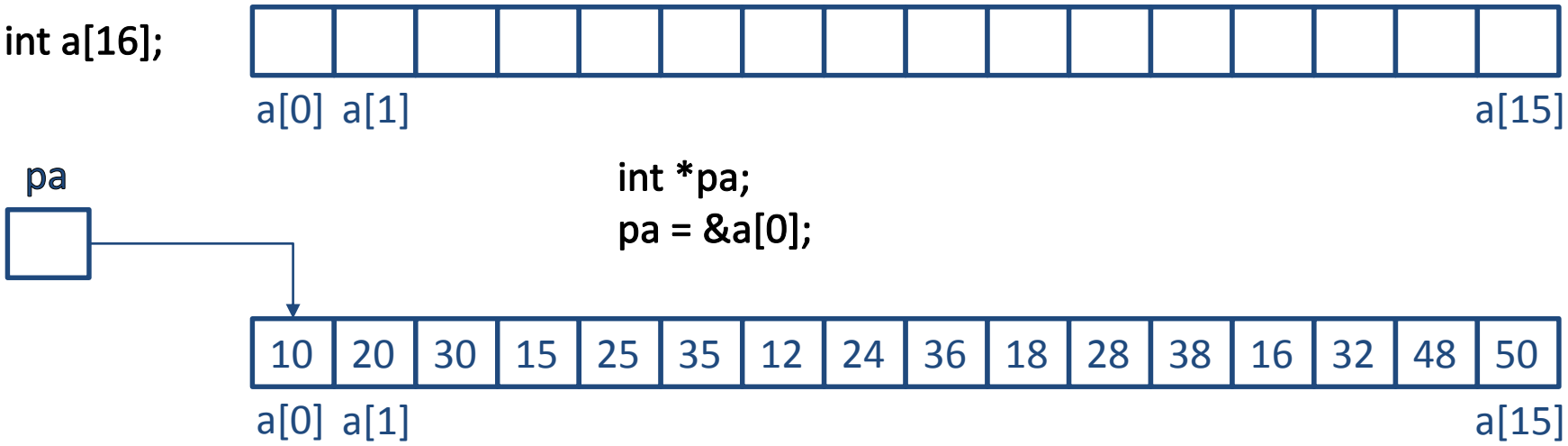
a[0] a[1] a[15]

```
int *pa;
pa = &a[0];
```

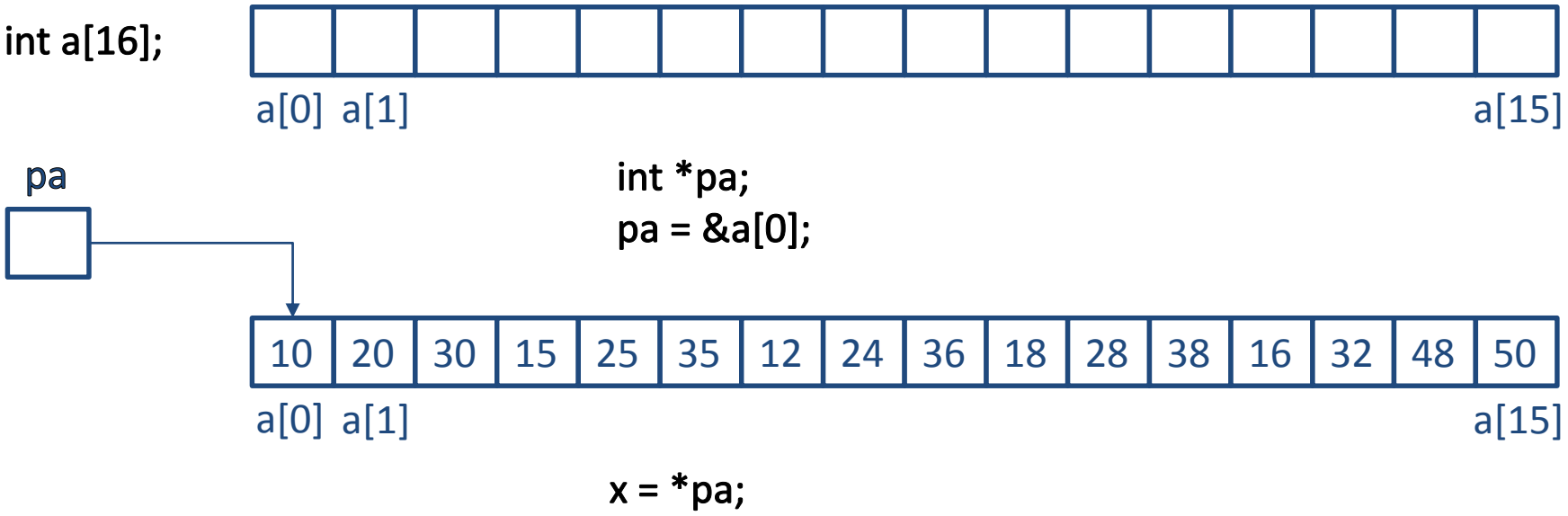
Pointers and Arrays



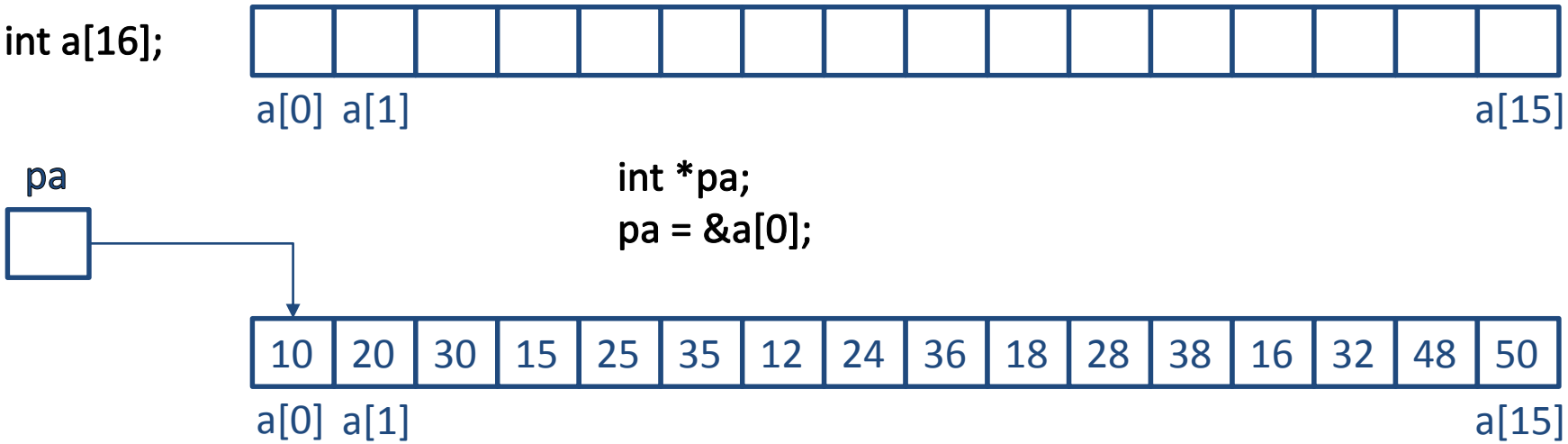
Pointers and Arrays



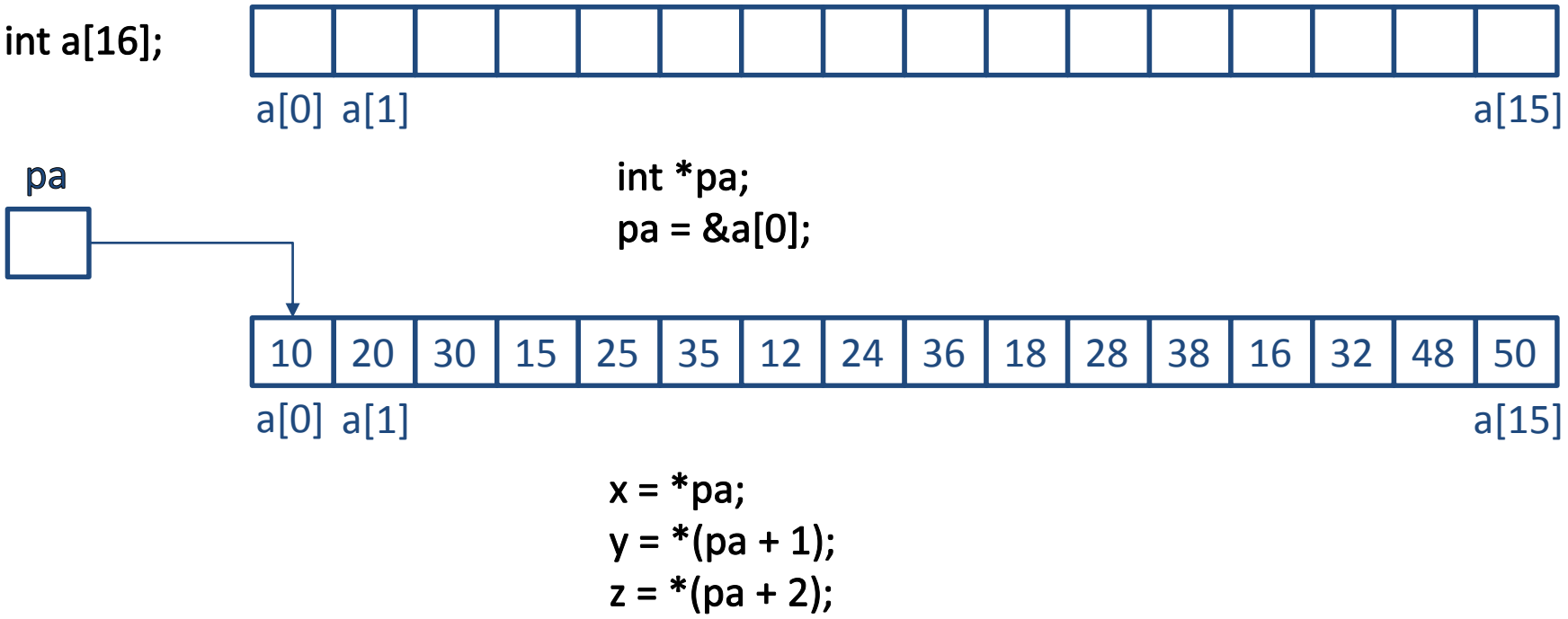
Pointers and Arrays



Pointers and Arrays

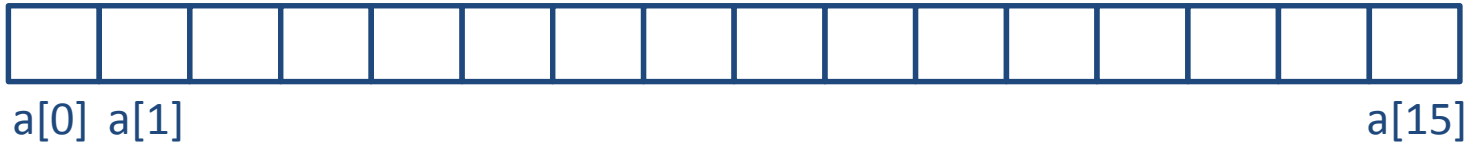


Pointers and Arrays



Pointers and Arrays

int a[16];

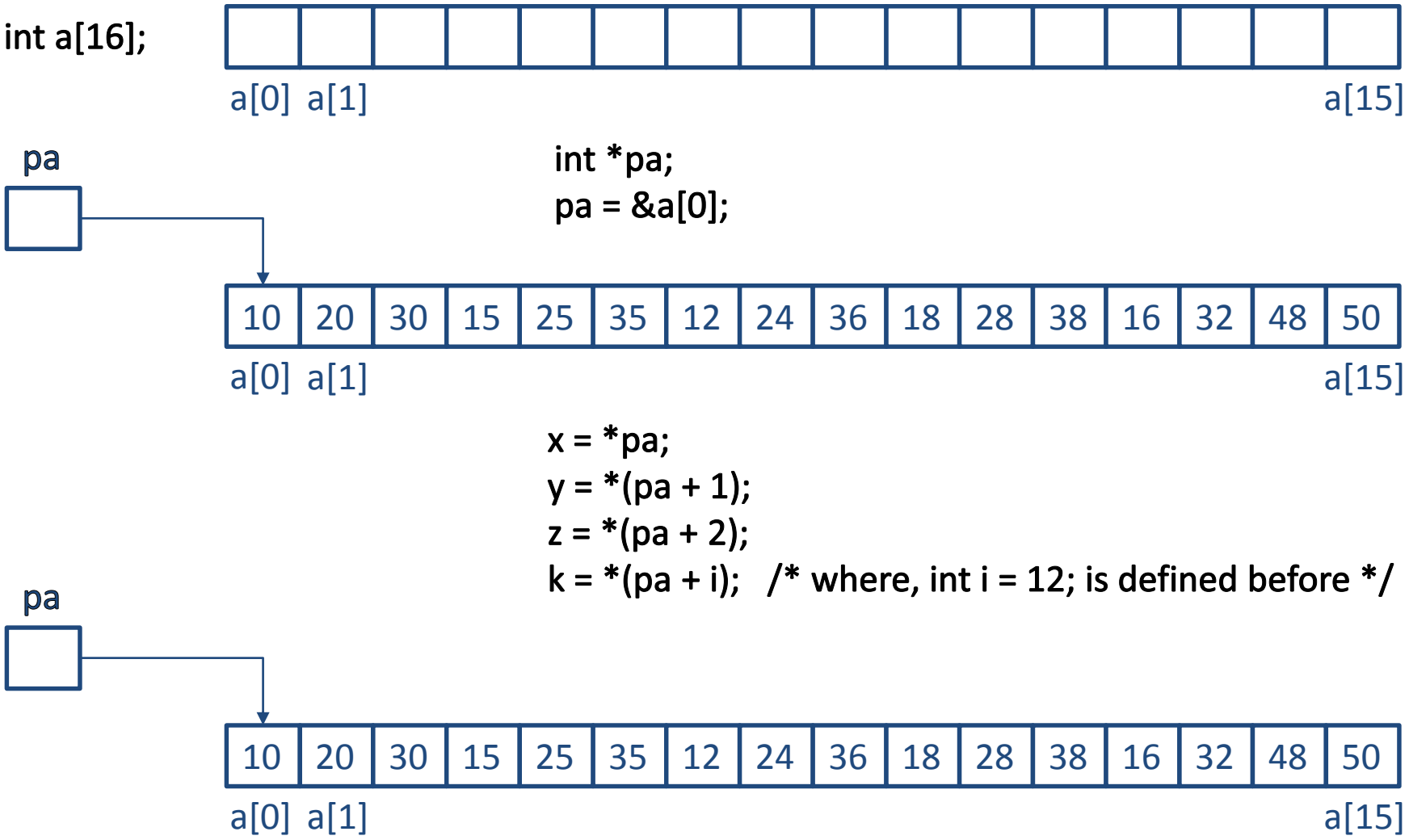


pa

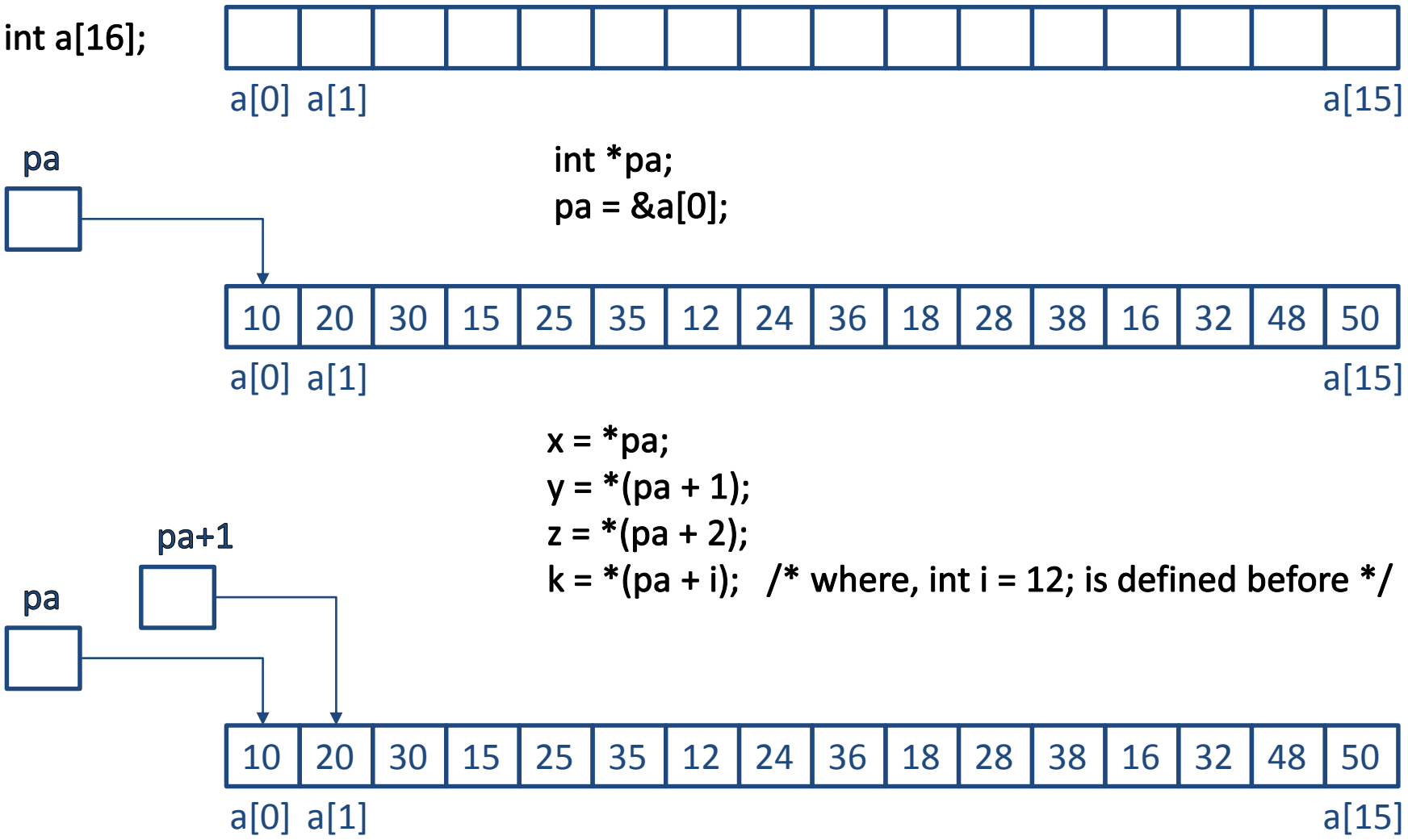
```
int *pa;
pa = &a[0];
```


```
x = *pa;
y = *(pa + 1);
z = *(pa + 2);
k = *(pa + i);   /* where, int i = 12; is defined before */
```

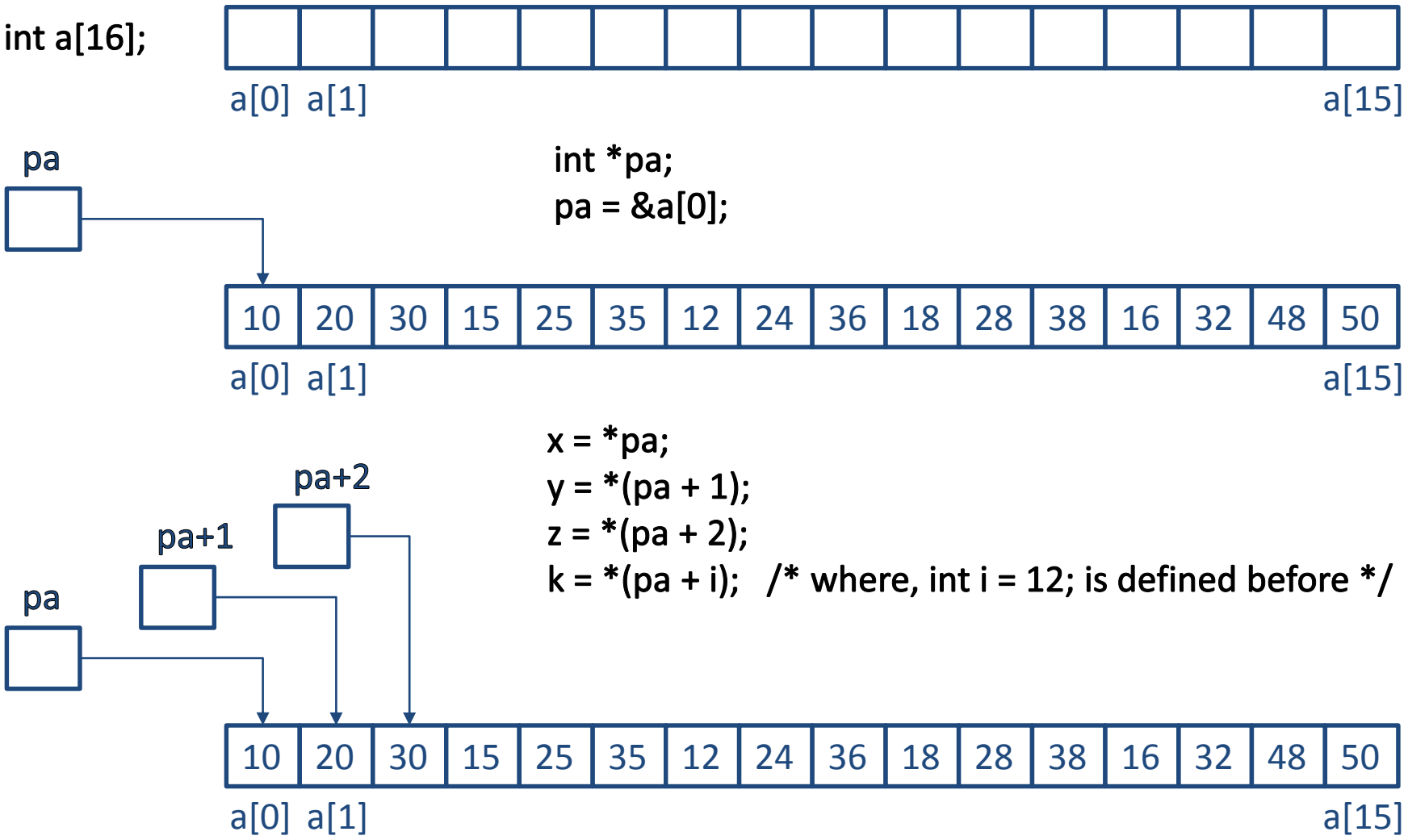
Pointers and Arrays



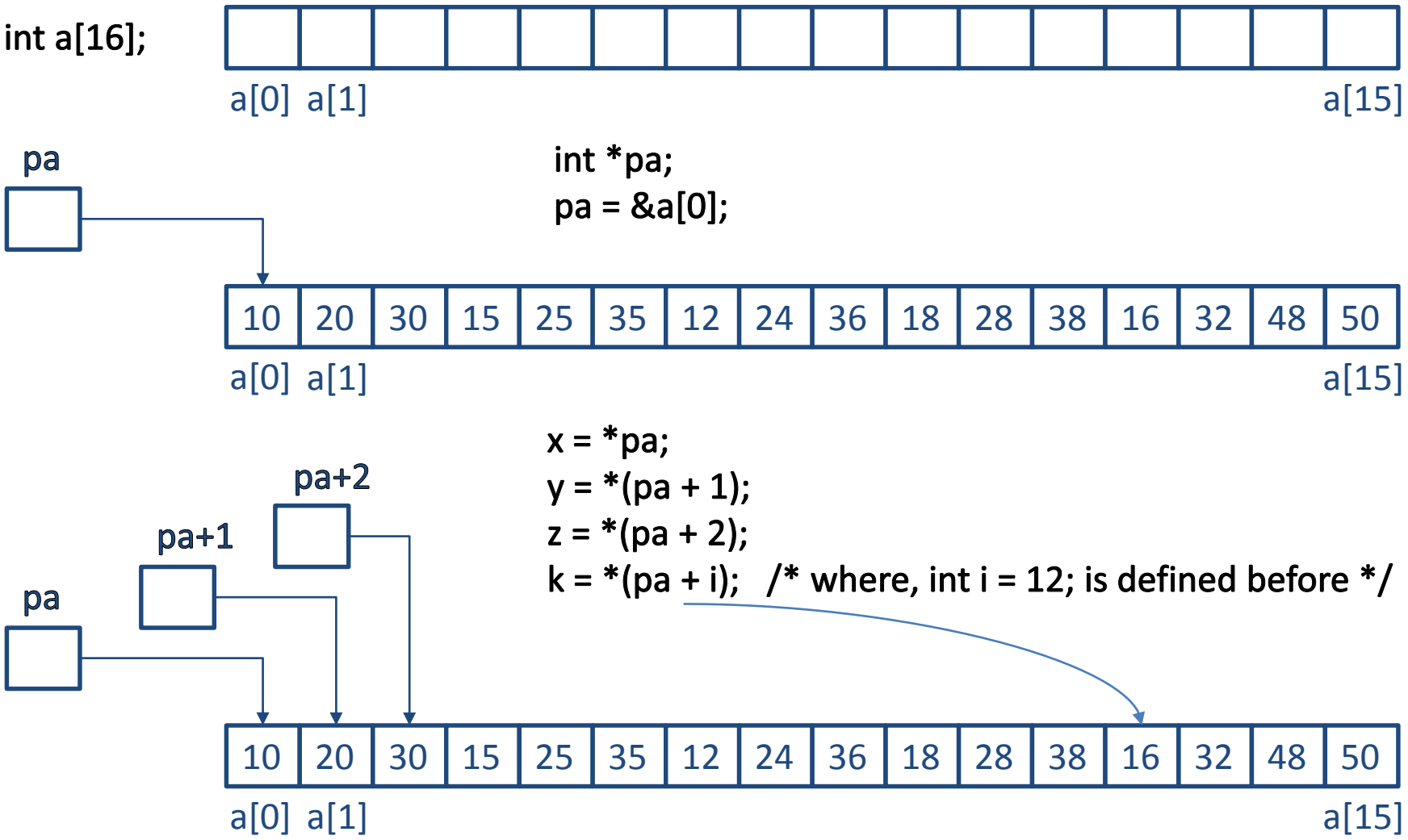
Pointers and Arrays



Pointers and Arrays



Pointers and Arrays



Pointer Arithmetic

If **p** is a pointer to some element of an array, then **p++** increments **p** to point to the next element, and **p += i** increments it to point **i** elements beyond where it currently does.

Dynamic Memory Allocation

- Pointer declaration does not allocate memory for values.

```
int *ptr;
```

- Use function **malloc** to allocate memory

```
malloc(sizeof(int))
```

  - Allocates number of bytes defined by the parameter
  - Memory allocated in the heap (not stack)
  - Returns a pointer to the block allocated
  - Memory allocated by malloc could be used to store any value
  - What should be the return type? (type of the pointer)

```
ptr = (int*)malloc(sizeof(int));
```

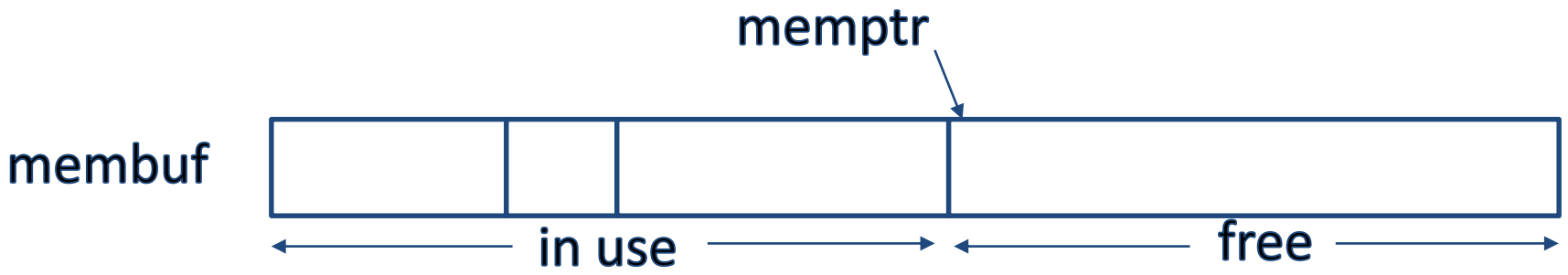
Dynamic Memory Allocation

We will develop our own simple allocator and call it *alloc()* and *afree()*.

Dynamic Memory Allocation

We will develop our own simple allocator and call it *alloc()* and *afree()*.

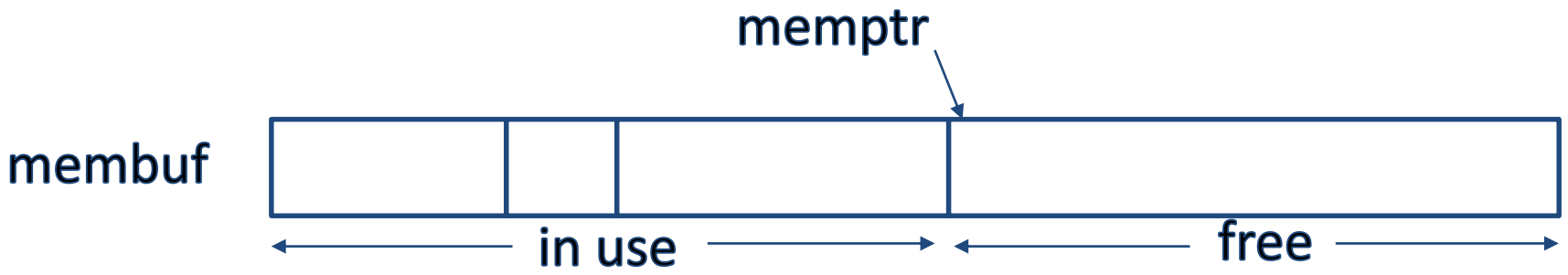
Before call to alloc:



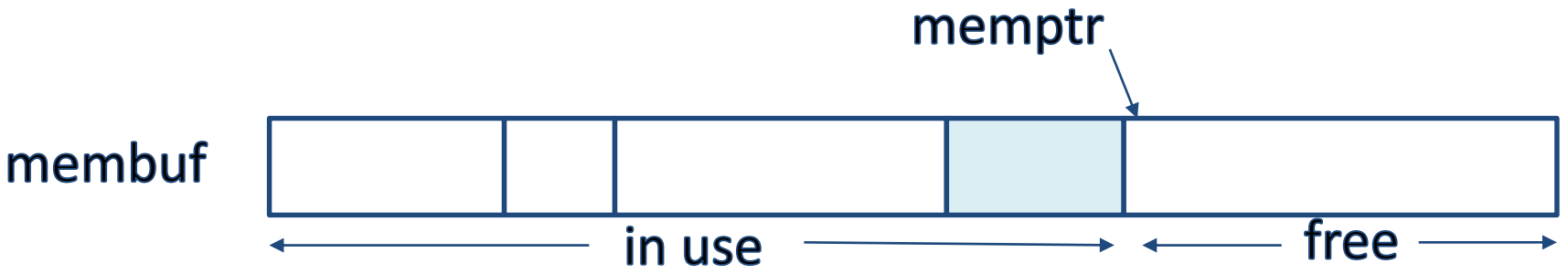
Dynamic Memory Allocation

We will develop our own simple allocator and call it *alloc()* and *afree()*.

Before call to alloc:



After call to alloc:



Dynamic Memory Allocation

```
#define MEMSIZE       1024;

static char membuffer[MEMSIZE];
static char *memptr = membuffer;

char *alloc(int n)
{
      if (membuffer + MEMSIZE – memptr >= n) {
            memptr += n;
            return (memptr - n);
      }
      else
            return 0;
}
```

Dynamic Memory Allocation

```
#define MEMSIZE       1024;

static char membuffer[MEMSIZE];
static char *memptr = membuffer;

void afree(char *ptr)
{
      if (ptr >= membuffer && ptr < membuffer + MEMSIZE)
            memptr = ptr;
}
```

Valid pointer operations

- Assignment of pointers of the same type.
- Adding or subtracting a pointer and an integer.
- Subtracting or comparing two pointers to members of the same array.
- Assigning or comparing pointers to zero.

Pointers vs Multi-dimensional Arrays

```
int a[10][80]; /* 800 int-sized locations */
int *b[10];     /* 10 pointers; initialization must be done explicitly */
```

Pointers vs Multi-dimensional Arrays

```
int a[10][80]; /* 800 int-sized locations */
int *b[10];      /* 10 pointers; initialization must be done explicitly */

char *pname[] = { “Unknown month”, “Jan”, “Feb” }
```

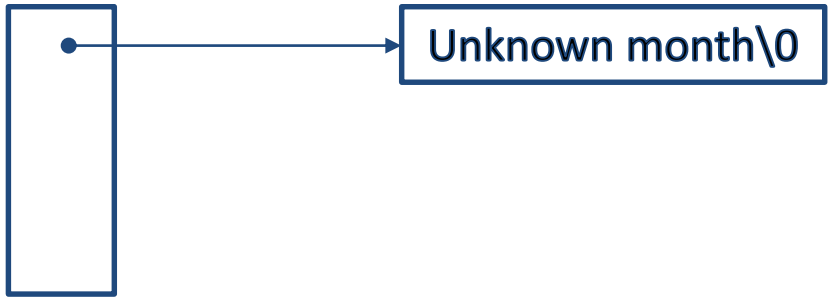
```
char aname[][14] = { “Unknown month”, “Jan”, “Feb” }
```

Pointers vs Multi-dimensional Arrays

int a[10][80]; /* 800 int-sized locations */

int *b[10]; /* 10 pointers; initialization must be done explicitly */

char *pname[] = { “Unknown month”, “Jan”, “Feb” }



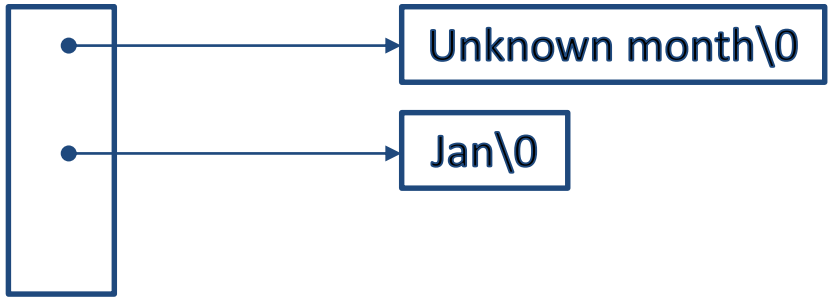
char aname[][14] = { “Unknown month”, “Jan”, “Feb” }

Pointers vs Multi-dimensional Arrays

int a[10][80]; /* 800 int-sized locations */

int *b[10]; /* 10 pointers; initialization must be done explicitly */

char *pname[] = { “Unknown month”, “Jan”, “Feb” }



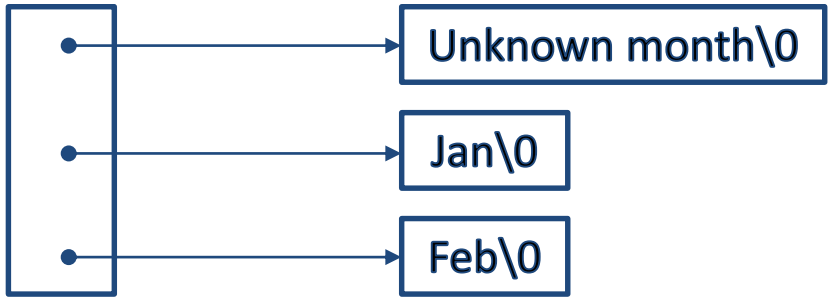
char aname[][14] = { “Unknown month”, “Jan”, “Feb” }

Pointers vs Multi-dimensional Arrays

int a[10][80]; /* 800 int-sized locations */

int *b[10]; /* 10 pointers; initialization must be done explicitly */

char *pname[] = { “Unknown month”, “Jan”, “Feb” }



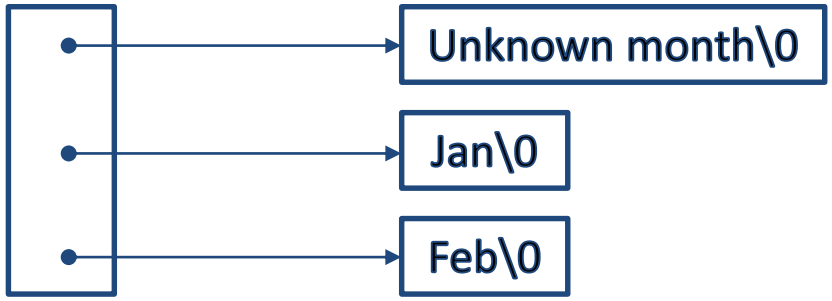
char aname[][14] = { “Unknown month”, “Jan”, “Feb” }

Pointers vs Multi-dimensional Arrays

int a[10][80]; /* 800 int-sized locations */

int *b[10]; /* 10 pointers; initialization must be done explicitly */

char *pname[] = { “Unknown month”, “Jan”, “Feb” }



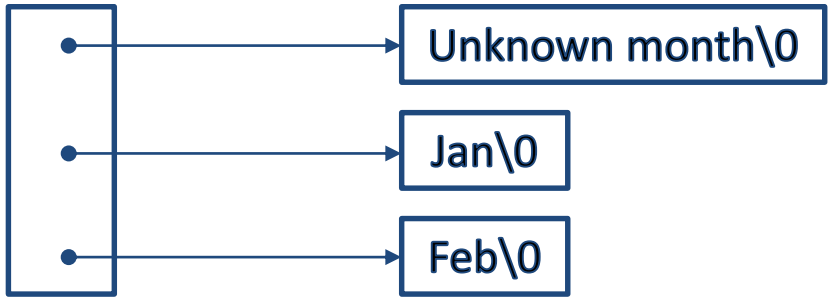
char aname[][14] = { “Unknown month”, “Jan”, “Feb” }

Pointers vs Multi-dimensional Arrays

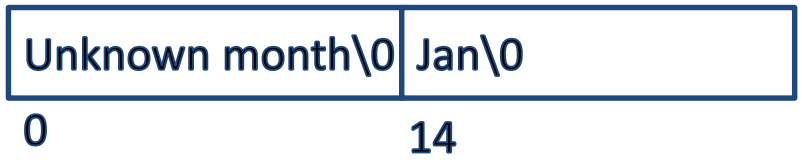
int a[10][80]; /* 800 int-sized locations */

int *b[10]; /* 10 pointers; initialization must be done explicitly */

char *pname[] = { “Unknown month”, “Jan”, “Feb” }



char aname[][14] = { “Unknown month”, “Jan”, “Feb” }

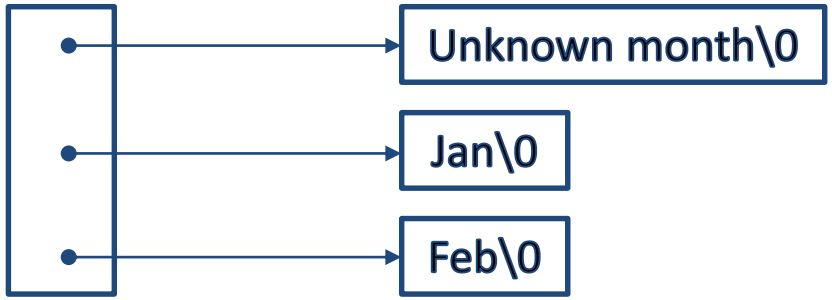


Pointers vs Multi-dimensional Arrays

int a[10][80]; /* 800 int-sized locations */

int *b[10]; /* 10 pointers; initialization must be done explicitly */

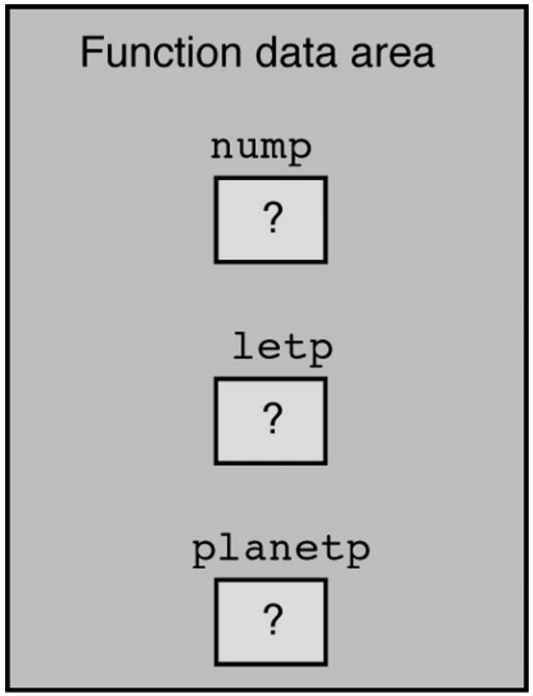
char *pname[] = { “Unknown month”, “Jan”, “Feb” }



char aname[][14] = { “Unknown month”, “Jan”, “Feb” }

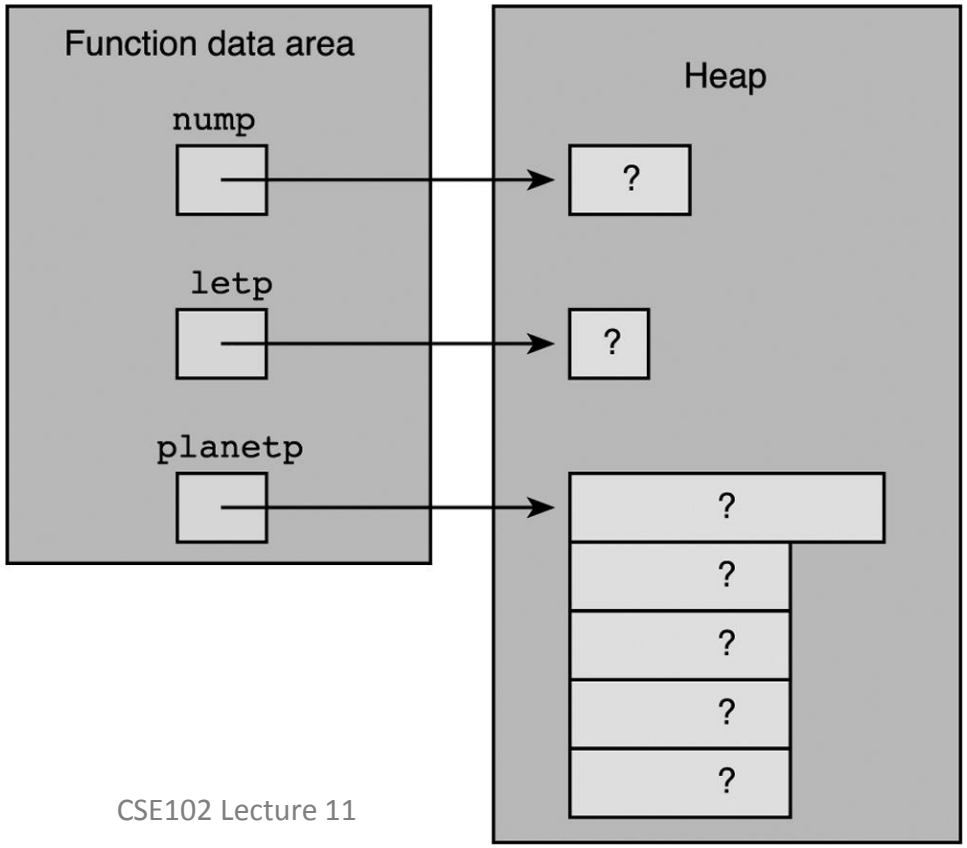
Three Pointer-Type Local Variables

```
int      *nump;
char     *letp;
planet_t *planetp;
```



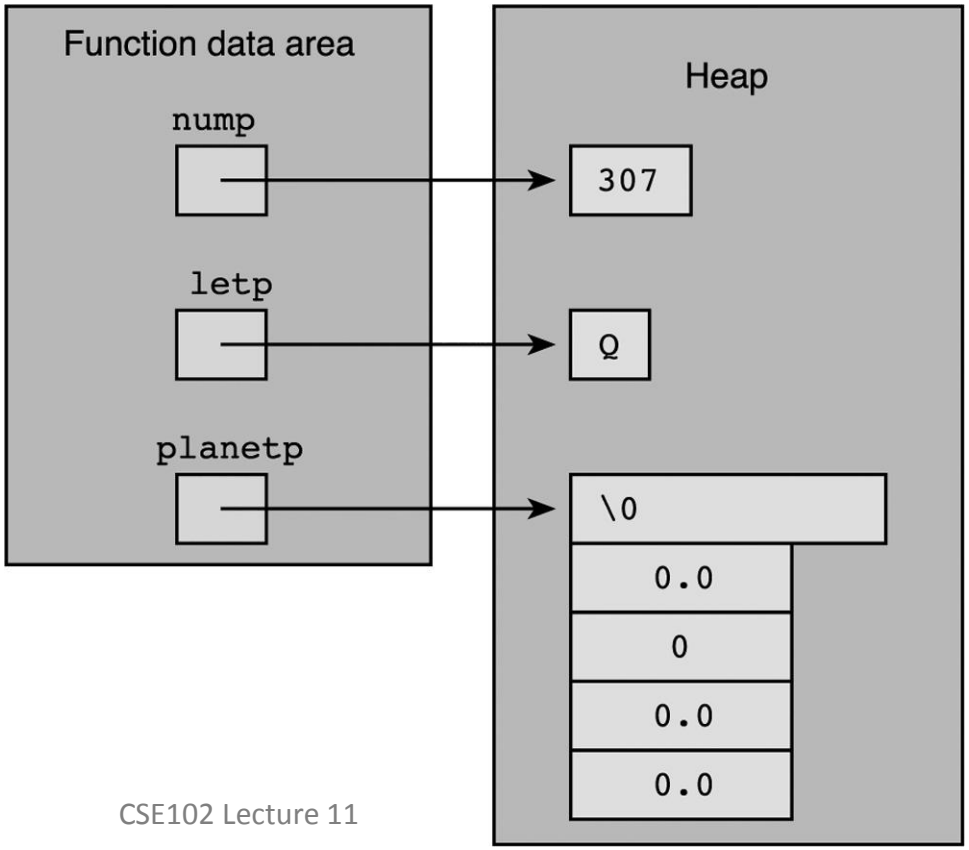
Dynamic Allocation of Variables

```
nump = (int *)malloc(sizeof(int));
letp = (char *)malloc(sizeof(char));
planetp = (planet_t *)malloc(sizeof(planet_t));
```



Assignments to Dynamically Allocated Variables

```
*nump = 307;
*letp = 'Q';
*planetp = blank_planet;
```



Components of Dynamically Allocated Structure

- Indirection + component selection

  `(*planetp).name`

- Indirect component selection

  `planetp->name`

```
1. printf("%s\n", planetp->name);
2. printf("  Equatorial diameter:  %.0f km\n", planetp->diameter);
3. printf("  Number of moons:  %d\n", planetp->moons);
4. printf("  Time to complete one orbit of the sun:  %.2f years\n",
5.        planetp->orbit_time);
6. printf("  Time to complete one rotation on axis:  %.4f hours\n",
7.        planetp->rotation_time);
```

Allocation of Arrays with calloc

- malloc: to allocate single memory block
- calloc: to dynamically create array of elements
  - Elements of any type (built-in or user defined)
- calloc: two arguments
  - Number of elements
  - Size of one element
- Allocates the memory and initializes to zero
- Returns a pointer

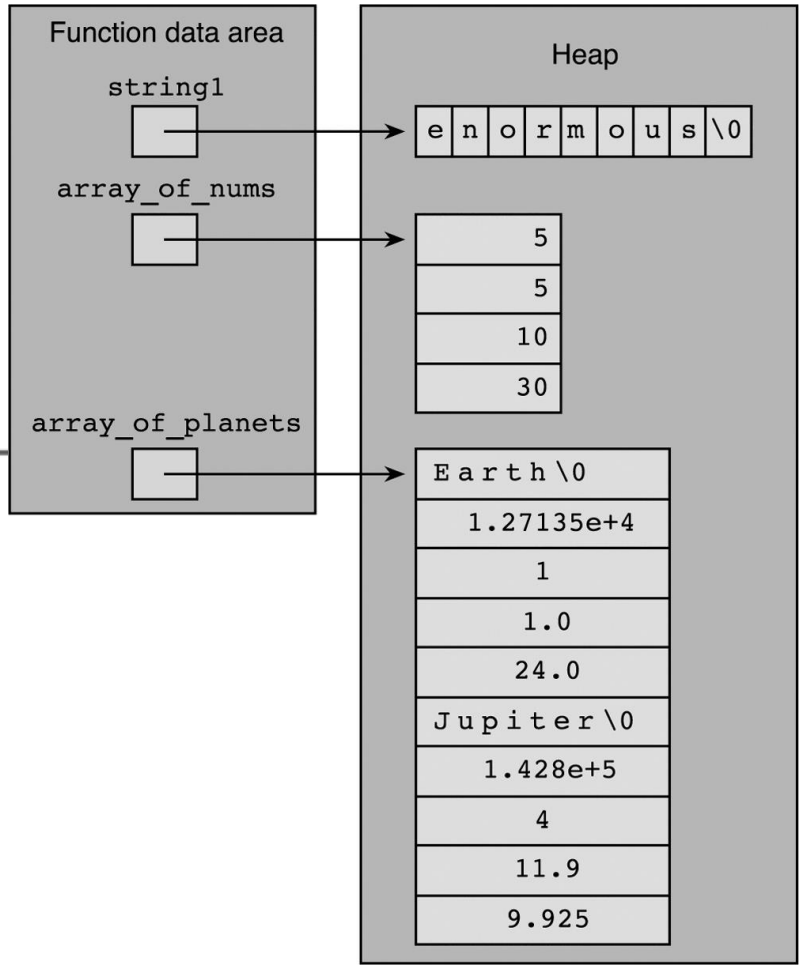
Allocation of Arrays with calloc

```
1.  #include <stdlib.h>  /* gives access to calloc */
2.  int scan_planet(planet_t *plnp);
3.
4.  int
5.  main(void)
6.  {
7.        char     *string1;
8.        int      *array_of_nums;
9.        planet_t *array_of_planets;
10.       int       str_siz, num_nums, num_planets, i;
11.       printf("Enter string length and string> ");
12.       scanf("%d", &str_siz);
13.       string1 = (char *)calloc(str_siz, sizeof (char));
14.       scanf("%s", string1);
15.
16.       printf("\nHow many numbers?> ");
17.       scanf("%d", &num_nums);
18.       array_of_nums = (int *)calloc(num_nums, sizeof (int));
19.       array_of_nums[0] = 5;
20.       for (i = 1;  i < num_nums;  ++i)
21.             array_of_nums[i] = array_of_nums[i - 1] * i;
22.
23.       printf("\nEnter number of planets and planet data> ");
24.       scanf("%d", &num_planets);
25.       array_of_planets = (planet_t *)calloc(num_planets,
26.                                             sizeof (planet_t));
```

(continued)

Allocation of Arrays with calloc

```
27.       for (i = 0;  i < num_planets;  ++i)
28.             scan_planet(&array_of_planets[i]);
29.       . . .
30. }

Enter string length and string> 9 enormous

How many numbers?> 4

Enter number of planets and planet data> 2
Earth    12713.5 1  1.0 24.0
Jupiter 142800.0 4 11.9  9.925
```



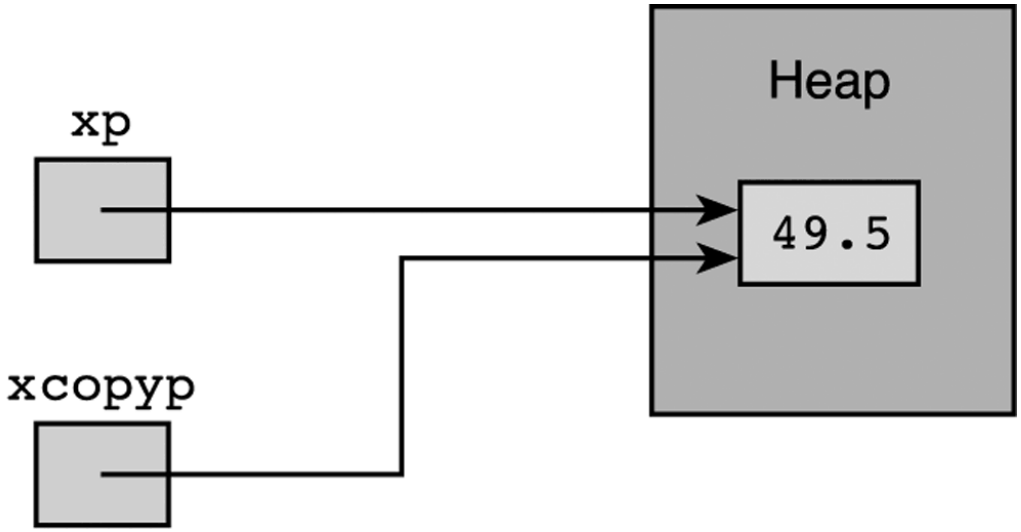
Returning Memory

- free : returns memory cells to heap
  - Allocated by calloc or malloc
  - Returned memory can be allocated later

```
free(nump);
free(array_of_planets);
```

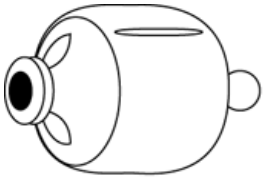
Multiple Pointers to a Cell

```
double *xp, *xcopyp;
xp = (double *)malloc(sizeof(double));
*xp = 49.5;
xcopyp = xp;
free(xp);
```

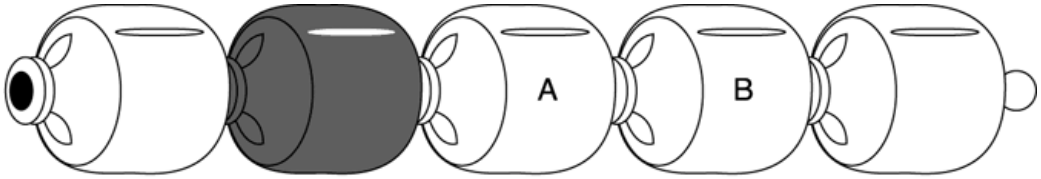


Linked Lists

- A sequence of nodes
  - Each node is connected to the following one
  - Like pop beads
    - A chain can be formed easily
    - Modified easily (remove and insert)



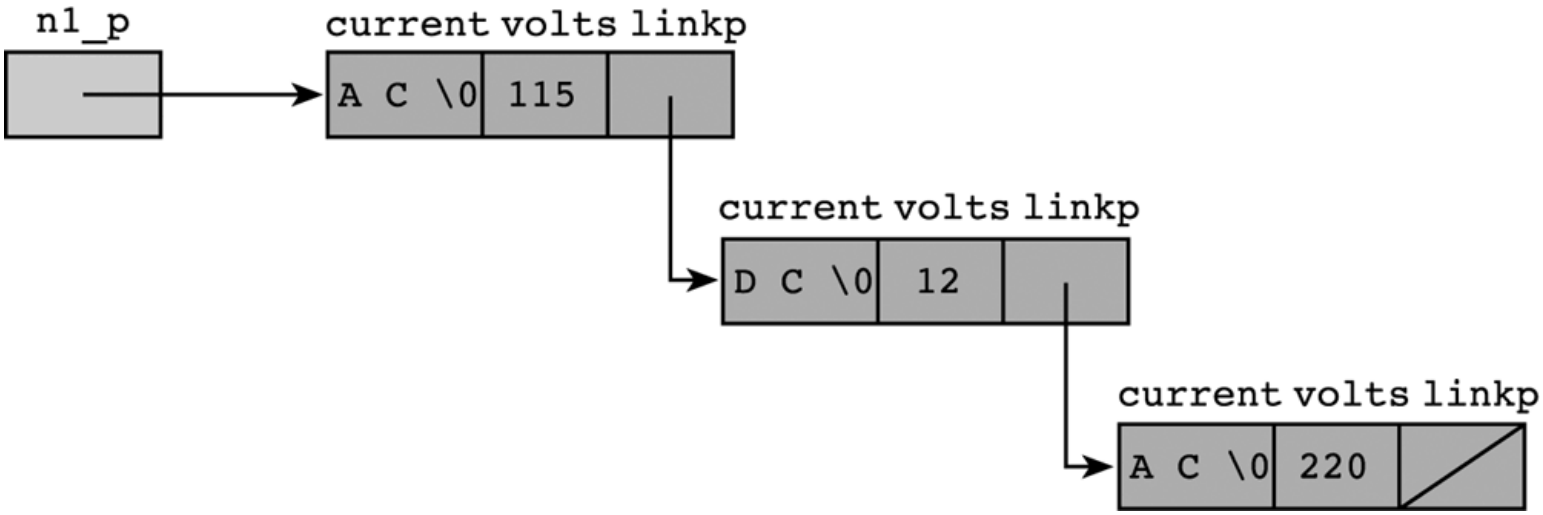
Pop bead



Chain of pop beads

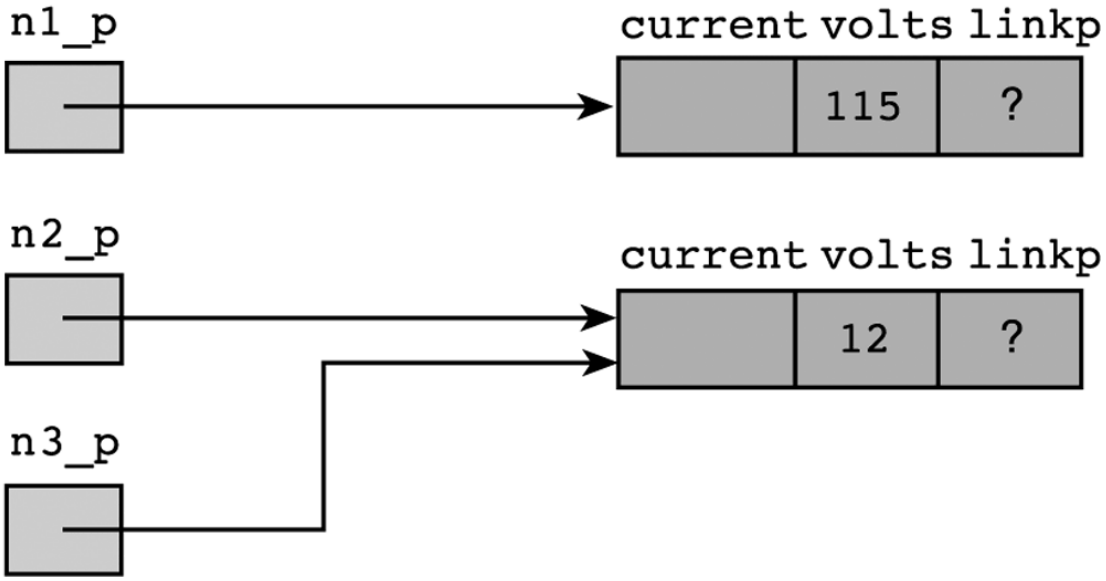
Node Structure

```
typedef struct node_s {
        char                current[3];
        int                 volts;
        struct node_s      *linkp;
} node_t;

node_t *n1_p;
```



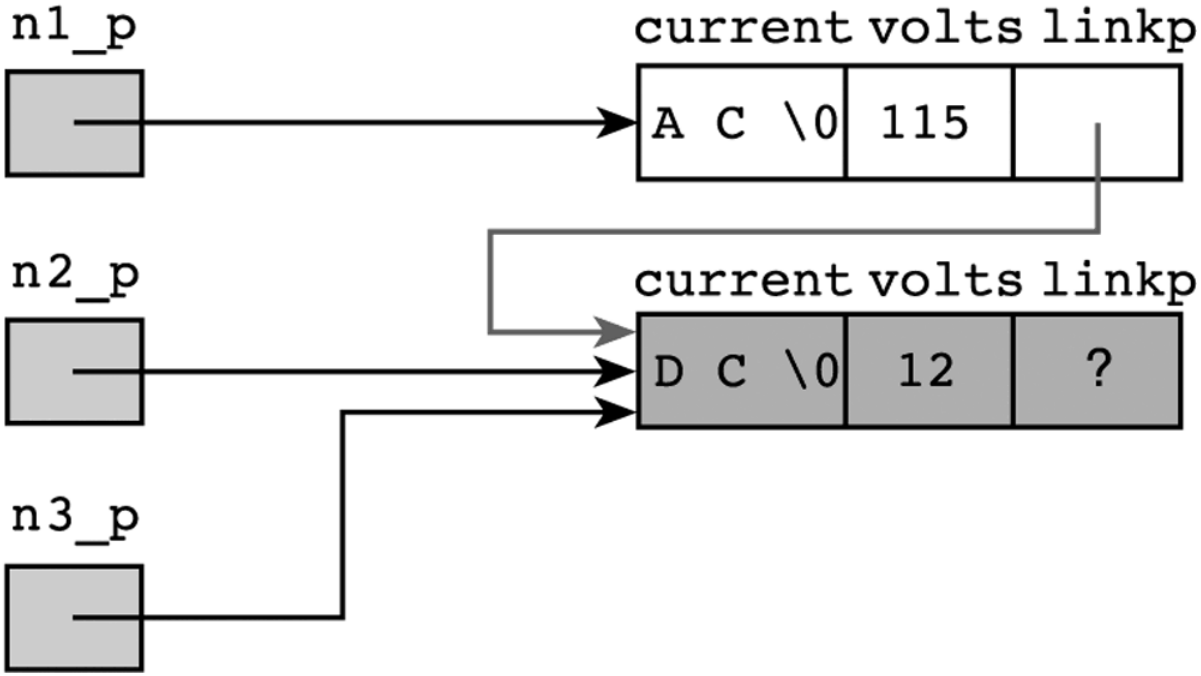
Creating Two Nodes

```
node_t *n1_p, *n2_p, *n3_p;
n1_p = (node_t *)malloc(sizeof(node_t));
n1_p->volts = 115;
n2_p = (node_t *)malloc(sizeof(node_t));
n2_p->volts = 12;
n3_p = n2_p;
```



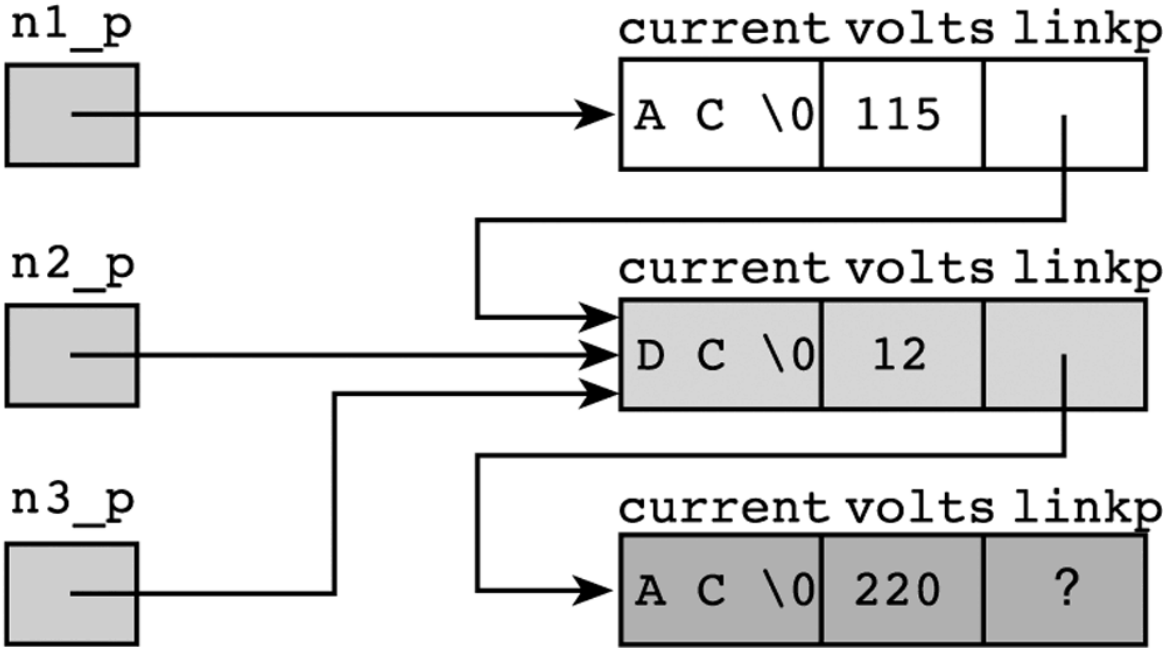
Linking Two Nodes

n1_p->linkp = n2_p;



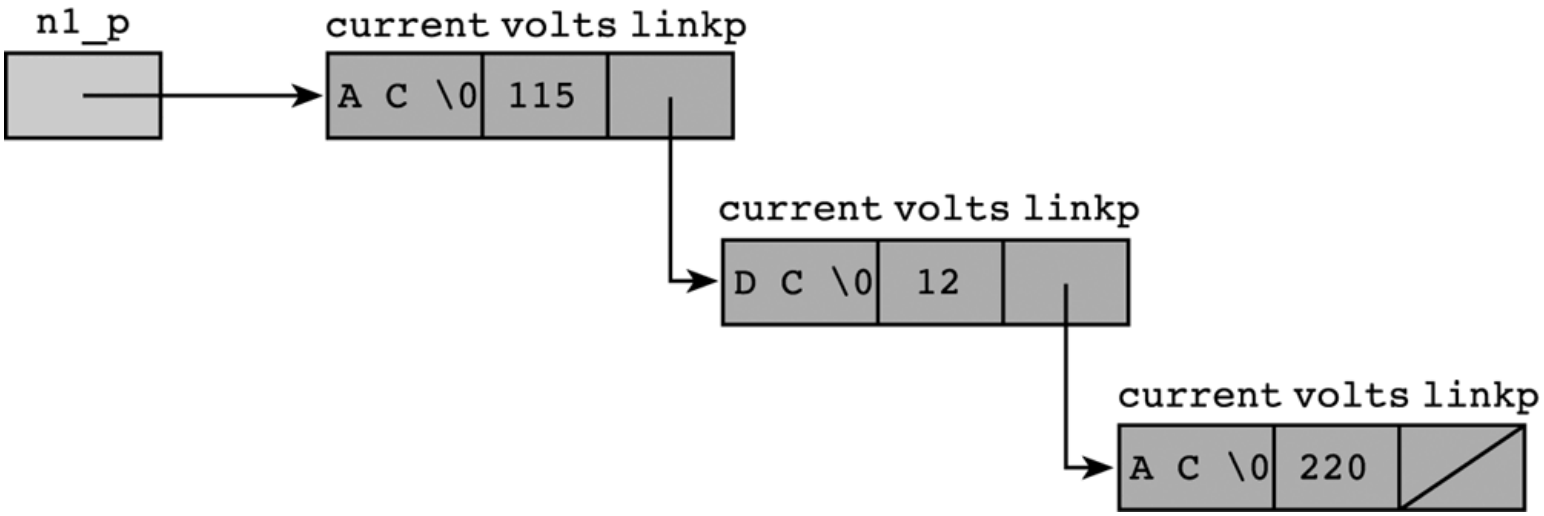
Three-Node Linked List

```
n2_p->linkp = (node_t *)malloc(sizeof(node)t));
n2_p->linkp->volts = 220;
strcpy(n2_p->linkp->current, “AC”);
```



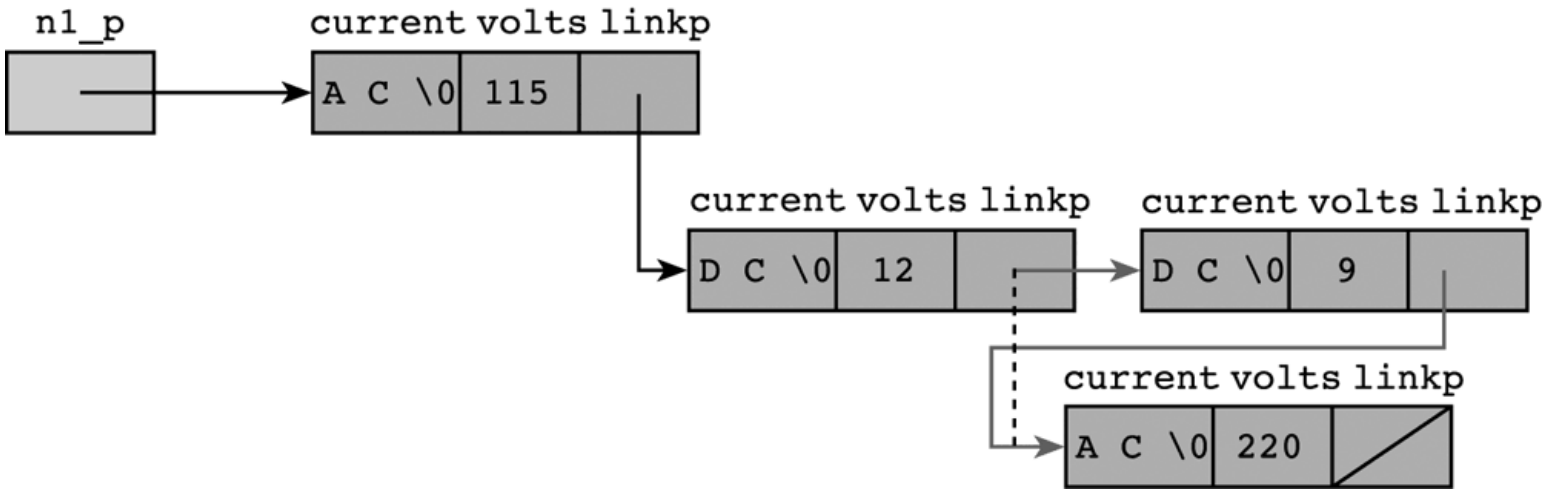
Three-Element Linked List

- List head
- End of list indicator
- Empty list



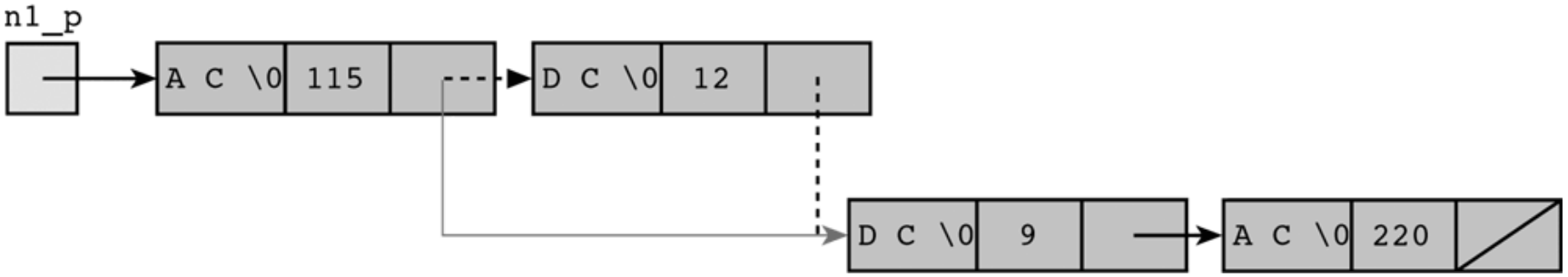
Linked List After an Insertion

- Linked List can be modified easily
  - Insertion



Linked List After a Deletion

- Linked List can be modified easily
  - Insertion
  - Deletion



Linked List Operations

- Assume following declaration

```
typedef struct list_node_s {
        int                     digit;
        struct list_node_s *restp;
} list_node_t;
```

- Traversing a list
  - Process each node in sequence
  - Start with the list head and follow list pointers
- Operations should take list head as a parameter

Recursive function print_list

```
1.  /*
2.   *  Displays the list pointed to by headp
3.   */
4.  void
5.  print_list(list_node_t *headp)
6.  {
7.       if (headp == NULL) {    /* simple case - an empty list                    */
8.            printf("\n");
9.       } else {                /* recursive step - handles first element         */
10.           printf("%d", headp->digit);  /*      leaves rest to                   */
11.           print_list(headp->restp);    /*      recursion                        */
12.      }
13. }
```

Recursive and Iterative List Printing

```
/* Displays the list pointed to by headp */
void
print_list(list_node_t *headp)
{
 if (headp == NULL) {/* simple case */
     printf("\n");
 } else {          /* recursive step */
     printf("%d", headp->digit);
     print_list(headp->restp);
 }
}
```

```
{ list_node_t *cur_nodep;

  for  (cur_nodep = headp; /* start at
                              beginning  */
        cur_nodep != NULL; /* not at
                              end yet    */
        cur_nodep = cur_nodep->restp)
      printf("%d", cur_nodep->digit);
  printf("\n");
 }
```

Node Structure (ll.h)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct
{
  char current[3];
  int volts;
  struct node *next;
} node;
```

Node Structure and Operations (ll.h)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct
{
  char current[3];
  int volts;
  struct node *next;
} node;

node *insert_front(char *current, int volts);
```

Node Structure and Operations (ll.h)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct
{
  char current[3];
  int volts;
  struct node *next;
} node;

node *insert_front(char *current, int volts);
node *insert_back(char *current, int volts);
```

Node Structure and Operations (ll.h)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct
{
  char current[3];
  int volts;
  struct node *next;
} node;

node *insert_front(char *current, int volts);
node *insert_back(char *current, int volts);
void delete_node(node *nd);
```

Node Structure and Operations (ll.h)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct
{
  char current[3];
  int volts;
  struct node *next;
} node;

node *insert_front(char *current, int volts);
node *insert_back(char *current, int volts);
void delete_node(node *nd);
void free_all();
```

Node Structure and Operations (ll.h)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct
{
  char current[3];
  int volts;
  struct node *next;
} node;

node *insert_front(char *current, int volts);
node *insert_back(char *current, int volts);
void delete_node(node *nd);
void free_all();
void print_list();
```

Node Operations (ll.c)

```
#include "ll.h"
node *head;

node *insert_front(char *current, int volts) {




}
```

Node Operations (ll.c)

```
#include "ll.h"
node *head;

node *insert_front(char *current, int volts) {
    node *n;
    n = (node *)malloc(sizeof(node));
    strcpy(n->current, current);
    n->volts = volts;

    if (head != 0) {
            n->next = (struct node *)head;
            head = n;
    }
    else {
            n->next = 0;
            head = n;
    }
    return n;
}
```

Node Operations (ll.c)

```
node *insert_back(char *current, int volts)
{




}
```

Node Operations (ll.c)

```
node *insert_back(char *current, int volts)
{
    node *n = head;
    node *prev = head;
    while (n != 0) {
           prev = n;
           n = (node *)n->next;
    }
    n = (node *)malloc(sizeof(node));
    strcpy(n->current, current);
    n->volts = volts;

    if (head != 0)
           prev->next = (struct node *)n;
    else
           head = n;
    n->next = 0;
    return n;
}
```

Node Operations (ll.c)

```
void delete_node(node *nd)
{

}
```

Node Operations (ll.c)

```
void delete_node(node *nd)
{
    node *n = head;
    node *prev = head;
    while (n != 0)
    {
            if (n == nd)
            {
                    if (n == head)
                            head = (node *)n->next;
                    prev->next = n->next;
                    free(n);
                    break;
            }
            prev = n;
            n = (node *)n->next;
    }
}
```

Node Operations (ll.c)

```
void free_all()
{









}
```

Node Operations (ll.c)

```
void free_all()
{
    node *n = head;
    while (n != 0)
    {
            head = (node *)n->next;
            free (n);
            n = head;
    }
}
```

Node Operations (ll.c)

```
void print_list()
{









}
```

Node Operations (ll.c)

```
void print_list()
{
    int i = 0;
    node *n = head;
    printf("Printing Node List:\n");
    while (n != 0)
    {
          printf("Node %5d --> Current: %3s Volts: %3d\n",
                  i, n->current, n->volts);
          n = (node *)n->next;
          i++;
    }
}
```

Node Operations (main.c)

```
int main() {

}
```

Node Operations (main.c)

```
int main() {
    node *n1  = insert_front("AC", 220);
    node *n2  = insert_front("DC", 110);
    node *n3  = insert_front("DC", 220);
    node *n4  = insert_front("AC", 220);
    node *n5  = insert_front("DC", 220);
    node *n6  = insert_front("DC", 120);
    node *n11 = insert_back("AC", 210);
    node *n12 = insert_back("DC", 120);
    node *n13 = insert_back("DC", 210);
    node *n14 = insert_back("AC", 210);
    node *n15 = insert_back("DC", 210);
    node *n16 = insert_back("DC", 110);

    print_list();
    delete_node(n1);
    print_list();
    free_all();
}
```

Directories and Files

```
salam@SALAM-PC: ls -l
drwx------+ 1 salam None    0 Dec  3 14:11 bin/
-rw-------+ 1 salam None 1.4K Dec  3 14:11 Makefile
drwx------+ 1 salam None    0 Dec  3 14:11 source/
```

Directories and Files

```
salam@SALAM-PC: ls -l
drwx------+ 1 salam None    0 Dec  3 14:11 bin/
-rw-------+ 1 salam None 1.4K Dec  3 14:11 Makefile
drwx------+ 1 salam None    0 Dec  3 14:11 source/

salam@SALAM-PC: ls -l source
drwx------+ 1 salam None    0 Dec  2 16:51 include/
drwx------+ 1 salam None    0 Dec  3 14:11 ll/
-rw-------+ 1 salam None  769 Dec  2 18:50 main.c
```

Directories and Files

```
salam@SALAM-PC: ls -l
drwx------+ 1 salam None    0 Dec  3 14:11 bin/
-rw-------+ 1 salam None 1.4K Dec  3 14:11 Makefile
drwx------+ 1 salam None    0 Dec  3 14:11 source/

salam@SALAM-PC: ls -l source
drwx------+ 1 salam None    0 Dec  2 16:51 include/
drwx------+ 1 salam None    0 Dec  3 14:11 ll/
-rw-------+ 1 salam None  769 Dec  2 18:50 main.c

salam@SALAM-PC: ls -l source/ll
-rw-------+ 1 salam None 1.3K Dec  2 18:49 ll.c
-rw-------+ 1 salam None  307 Dec  2 18:47 ll.h
```

Directories and Files - Compiling and Linking

```
salam@SALAM-PC: ls -l
drwx------+ 1 salam None    0 Dec  3 14:11 bin/
-rw-------+ 1 salam None 1.4K Dec  3 14:11 Makefile
drwx------+ 1 salam None    0 Dec  3 14:11 source/

salam@SALAM-PC: ls -l source
drwx------+ 1 salam None    0 Dec  2 16:51 include/
drwx------+ 1 salam None    0 Dec  3 14:11 ll/
-rw-------+ 1 salam None  769 Dec  2 18:50 main.c

salam@SALAM-PC: ls -l source/ll
-rw-------+ 1 salam None 1.3K Dec  2 18:49 ll.c
-rw-------+ 1 salam None  307 Dec  2 18:47 ll.h

gcc -Ofast -ansi  -c source/ll/ll.c -o source/ll/ll.o
```

Directories and Files - Compiling and Linking

```
salam@SALAM-PC: ls -l
drwx------+ 1 salam None    0 Dec  3 14:11 bin/
-rw-------+ 1 salam None 1.4K Dec  3 14:11 Makefile
drwx------+ 1 salam None    0 Dec  3 14:11 source/

salam@SALAM-PC: ls -l source
drwx------+ 1 salam None    0 Dec  2 16:51 include/
drwx------+ 1 salam None    0 Dec  3 14:11 ll/
-rw-------+ 1 salam None  769 Dec  2 18:50 main.c

salam@SALAM-PC: ls -l source/ll
-rw-------+ 1 salam None 1.3K Dec  2 18:49 ll.c
-rw-------+ 1 salam None  307 Dec  2 18:47 ll.h

gcc -Ofast -ansi  -c source/ll/ll.c -o source/ll/ll.o

gcc -Ofast -ansi  -c source/main.c -o source/main.o
```

Directories and Files - Compiling and Linking

```
salam@SALAM-PC: ls -l
drwx------+ 1 salam None    0 Dec  3 14:11 bin/
-rw-------+ 1 salam None 1.4K Dec  3 14:11 Makefile
drwx------+ 1 salam None    0 Dec  3 14:11 source/

salam@SALAM-PC: ls -l source
drwx------+ 1 salam None    0 Dec  2 16:51 include/
drwx------+ 1 salam None    0 Dec  3 14:11 ll/
-rw-------+ 1 salam None  769 Dec  2 18:50 main.c

salam@SALAM-PC: ls -l source/ll
-rw-------+ 1 salam None 1.3K Dec  2 18:49 ll.c
-rw-------+ 1 salam None  307 Dec  2 18:47 ll.h

gcc -Ofast -ansi  -c source/ll/ll.c -o source/ll/ll.o

gcc -Ofast -ansi  -c source/main.c -o source/main.o

gcc -Ofast -ansi  source/ll/ll.o source/main.o  -o bin/ll.exe
```

Makefile

```
ROOT  := source

BIN_DIR         := bin
INCLUDES_DIR    := $(ROOT)/include
LL_DIR          := $(ROOT)/ll
MAIN_DIR        := $(ROOT)

CC        := gcc
LD        := gcc
CFLAGS    := -Ofast -ansi
LDFLAGS   := -Ofast -ansi

SRCS       =      $(LL_DIR)/ll.c \
                  $(MAIN_DIR)/main.c

all: $(SRCS) ll.exe

OBJS      := $(SRCS:.c=.o)
```

Makefile

```
ll.exe: $(OBJS)
    @printf   "   LD                $(BIN_DIR)/$@\n"
    $(LD) $(LDFLAGS) $(OBJS) $(LIBS) -o $(BIN_DIR)/$@

.c.o:
    @printf   "   CC                $@\n"
    $(CC) $(CFLAGS) -c $< -o $@

clean:
    @printf   "   RM \n $(OBJS)\n"
    $(RM) $(OBJS)

clean_all:
    @printf   "   RM \n $(OBJS) $(BIN_DIR)/ll.exe\n"
    $(RM) $(OBJS) $(BIN_DIR)/ll.exe
```

Directories and Files – Building ll.exe

```
salam@SALAM-PC: ls -l
drwx------+ 1 salam None    0 Dec  3 14:11 bin/
-rw-------+ 1 salam None 1.4K Dec  3 14:11 Makefile
drwx------+ 1 salam None    0 Dec  3 14:11 source/

salam@SALAM-PC: ls -l source
drwx------+ 1 salam None    0 Dec  2 16:51 include/
drwx------+ 1 salam None    0 Dec  3 14:11 ll/
-rw-------+ 1 salam None  769 Dec  2 18:50 main.c

salam@SALAM-PC: ls -l source/ll
-rw-------+ 1 salam None 1.3K Dec  2 18:49 ll.c
-rw-------+ 1 salam None  307 Dec  2 18:47 ll.h

salam@SALAM-PC: make all
   CC              source/ll/ll.o
gcc -Ofast -ansi -c source/ll/ll.c -o source/ll/ll.o
   CC              source/main.o
gcc -Ofast -ansi -c source/main.c -o source/main.o
   LD              bin/ll.exe
gcc -Ofast -ansi source/ll/ll.o source/main.o -o bin/ll.exe
```

Linked List Representation of Stacks

Linked List Representation of Stacks

- Stack can be implemented using array

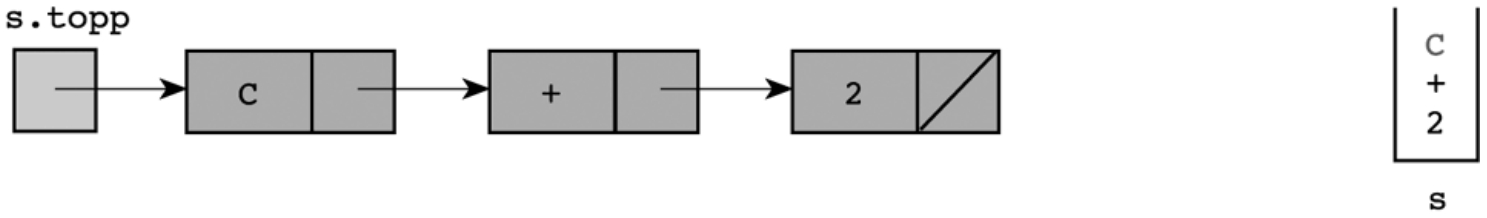
Linked List Representation of Stacks

- Stack can be implemented using array
- Stack = LIFO List
  - Linked list can be used

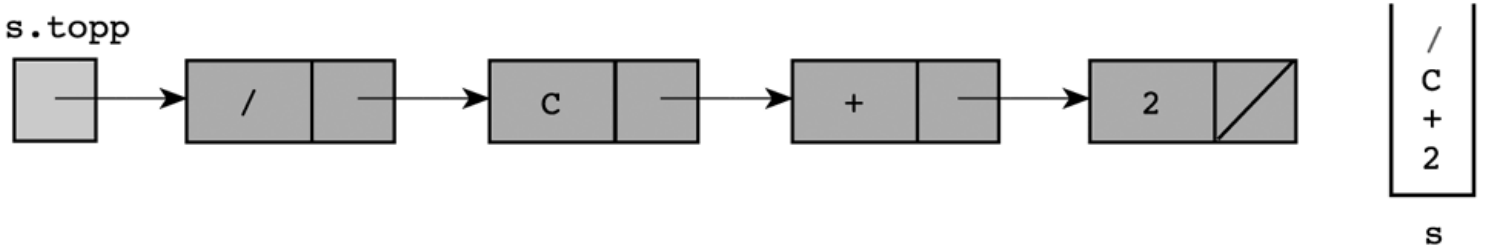
Linked List Representation of Stacks

- Stack can be implemented using array
- Stack = LIFO List
  - Linked list can be used

Stack of three characters



Stack after insertion (push) of '/'



Stack Structure

```
1.  typedef char stack_element_t;
2.
3.  typedef struct stack_node_s {
4.        stack_element_t      element;
5.        struct stack_node_s *restp;
6.  } stack_node_t;
7.
8.  typedef struct {
9.        stack_node_t *topp;
10. } stack_t;
```

Stack Manipulation with push and pop

```
1.  /*
2.   *  Creates and manipulates a stack of characters
3.   */
4.
5.  #include <stdio.h>
6.  #include <stdlib.h>
7.
8.  /*  Include typedefs from Fig. 14.24 */
9.  void push(stack_t *sp, stack_element_t c);
10. stack_element_t pop(stack_t *sp);
```

(continued)

Stack Manipulation with push and pop

```
#include <stdio.h>
#include <stdlib.h>

/*  Include typedefs from Fig. 14.24 */
void push(stack_t *sp, stack_element_t c);
stack_element_t pop(stack_t *sp);
int
main(void)
{
      stack_t s = {NULL};  /* stack of characters - initially empty */

      /*  Builds first stack of Fig. 14.23        */
      push(&s, '2');
      push(&s, '+');
      push(&s, 'C');

      /*  Completes second stack of Fig. 14.23    */
      push(&s, '/');

      /*  Empties stack element by element        */
      printf("\nEmptying stack: \n");
      while (s.topp != NULL) {
            printf("%c\n", pop(&s));
      }

      return (0);
}
```

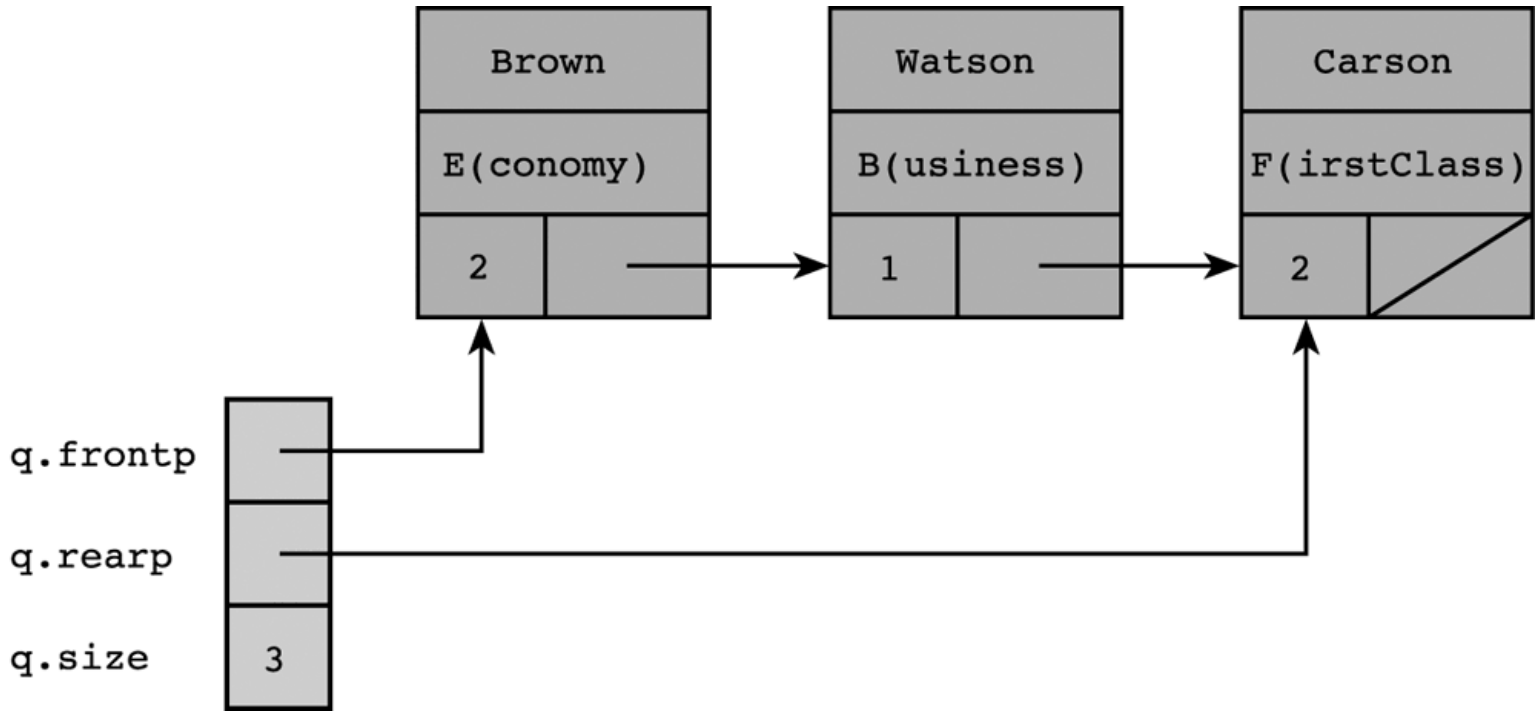
```
33. /*
34.  *  The value in c is placed on top of the stack accessed through sp
35.  *  Pre:  the stack is defined
36.  */
37. void
38. push(stack_t         *sp, /* input/output - stack            */
39.      stack_element_t c)  /* input         - element to add */
40. {
41.       stack_node_t *newp;  /* pointer to new stack node  */
42.
43.       /*  Creates and defines new node                    */
44.       newp = (stack_node_t *)malloc(sizeof (stack_node_t));
45.       newp->element = c;
46.       newp->restp = sp->topp;
47.       /*  Sets stack pointer to point to new node       */
48.       sp->topp = newp;
49. }
```

```
51. /*
52.  *  Removes and frees top node of stack, returning character value
53.  *  stored there.
54.  *  Pre:  the stack is not empty
55.  */
56. stack_element_t
57. pop(stack_t *sp) /* input/output - stack */
58. {
59.       stack_node_t    *to_freep;  /* pointer to node removed */
60.       stack_element_t  ans;       /* value at top of stack   */
61.
62.       to_freep = sp->topp;           /* saves pointer to node being deleted     */
63.       ans = to_freep->element;       /* retrieves value to return               */
64.       sp->topp = to_freep->restp;    /* deletes top node                        */
65.       free(to_freep);                /* deallocates space                       */
66.
67.       return (ans);
68. }
```

Queue

- Queue = FIFO List
- EX: Model a line of customers waiting at a checkout counter
- Need to keep both ends of a queue
  - Front and rear

A Queue of Passengers



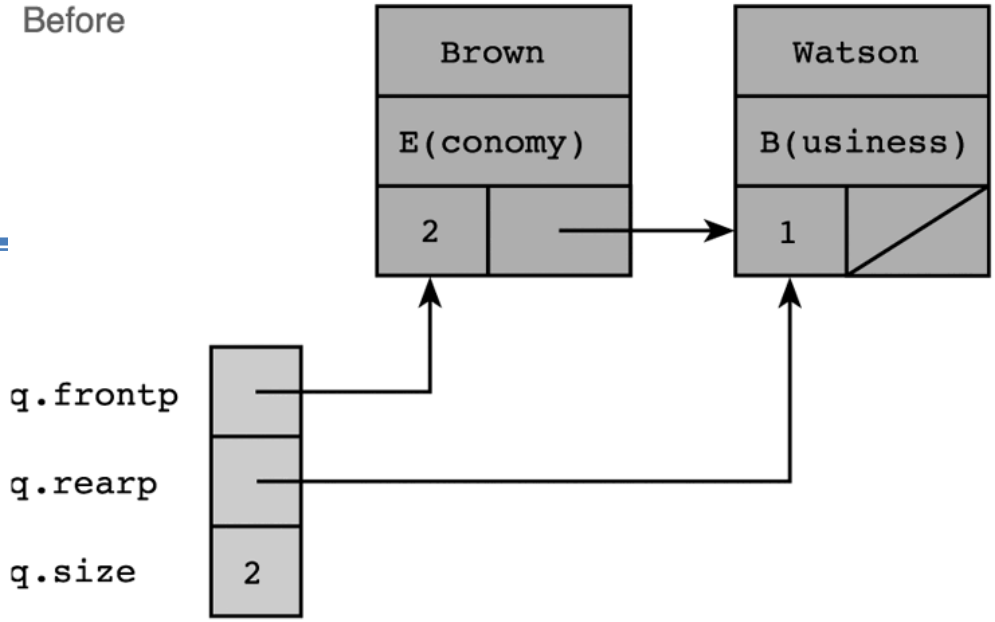
Queue Structure

```
1.  /*  Insert typedef for queue_element_t */
2.
3.  typedef struct queue_node_s {
4.        queue_element_t       element;
5.        struct queue_node_s *restp;
6.  } queue_node_t;
7.
8.  typedef struct {
9.       queue_node_t *frontp,
10.                   *rearp;
11.       int            size;
12. } queue_t;
```

```
1.  /*
2.   *  Creates and manipulates a queue of passengers.
3.   */
4.
5.  int scan_passenger(queue_element_t *passp);
6.  void print_passenger(queue_element_t pass);
7.  void add_to_q(queue_t *qp, queue_element_t ele);
8.  queue_element_t remove_from_q(queue_t *qp);
9.  void display_q(queue_t q);
10.
11. int
12. main(void)
13. {
14.       queue_t pass_q = {NULL, NULL, 0}; /* passenger queue - initialized to
15.                                            empty state */
16.       queue_element_t next_pass, fst_pass;
17.       char choice;  /* user's request */
18.
19.       /*  Processes requests  */
20.       do {
21.           printf("Enter A(dd), R(emove), D(isplay), or Q(uit)> ");
22.           scanf(" %c", &choice);
23.           switch (toupper(choice)) {
24.           case 'A':
25.                 printf("Enter passenger data> ");
26.                 scan_passenger(&next_pass);
27.                 add_to_q(&pass_q, next_pass);
28.                 break;
29.
```

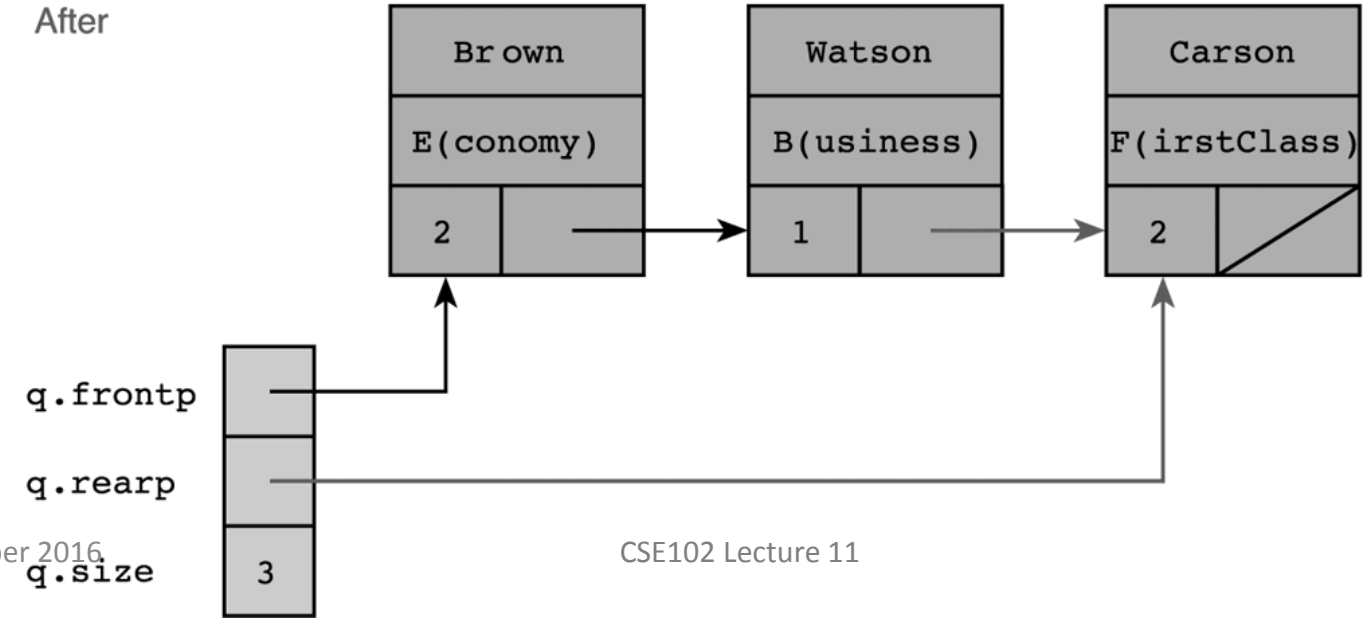
```
30.         case 'R':
31.               if (pass_q.size > 0) {
32.                     fst_pass = remove_from_q(&pass_q);
33.                     printf("Passenger removed from queue: \n");
34.                     print_passenger(fst_pass);
35.               } else {
36.                     printf("Queue empty - noone to delete\n");
37.               }
38.               break;
39.
40.         case 'D':
41.               if (pass_q.size > 0)
42.                     display_q(pass_q);
43.               else
44.                     printf("Queue is empty\n");
45.               break;
46.
47.         case 'Q':
48.               printf("Leaving passenger queue program with %d \n",
49.                      pass_q.size);
50.               printf("passengers in the queue\n");
51.               break;
52.
53.         default:
54.               printf("Invalid choice -- try again\n");
55.         }
56.      } while (toupper(choice) != 'Q');
57.
58.      return (0);
59. }
```

Before

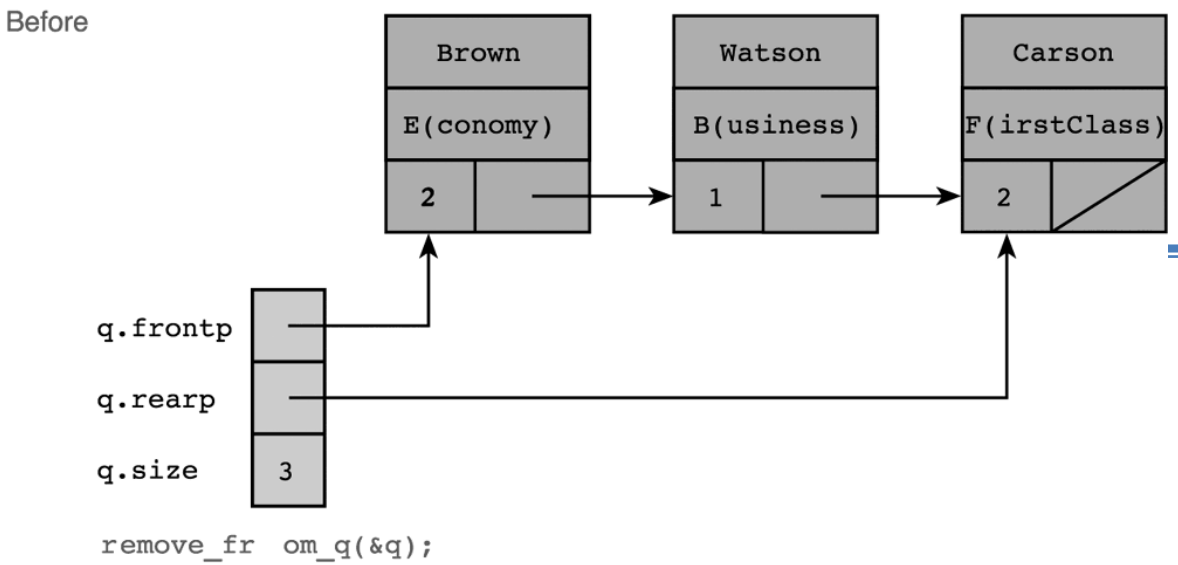


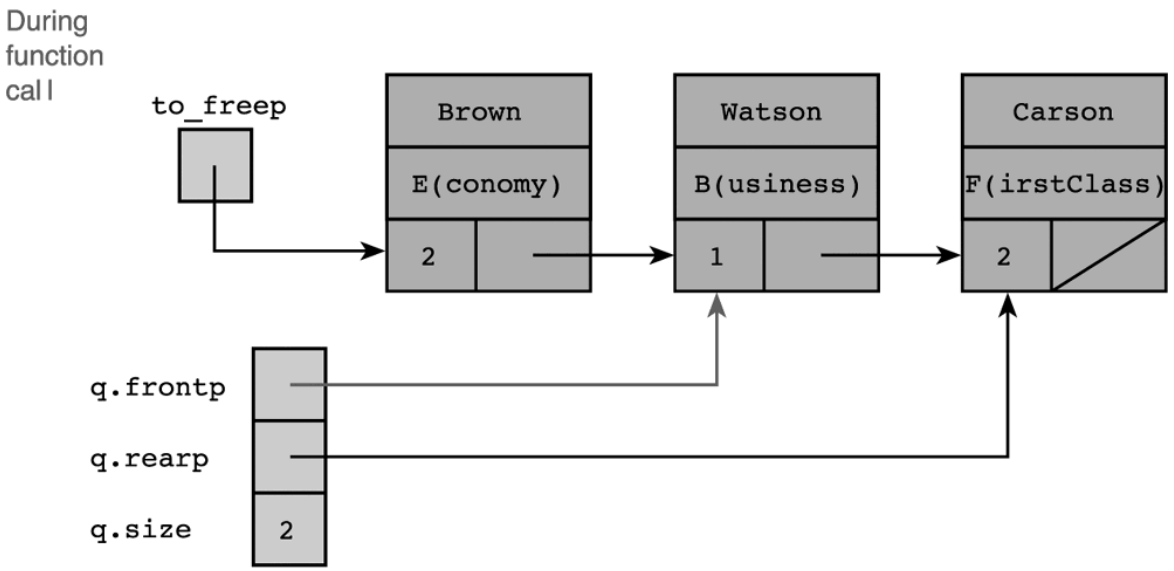

```
add_to_q(&q, next_pass);
```

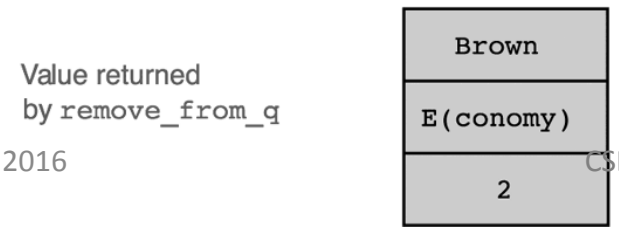
After



```
1.  /*
2.   *  Adds ele at the end of queue accessed through qp
3.   *  Pre:  queue is not empty
4.   */
5.  void
6.  add_to_q(queue_t          *qp,  /* input/output - queue   */
7.           queue_element_t  ele) /* input - element to add */
8.  {
9.        if (qp->size == 0) {                  /* adds to empty queue                */
10.             qp->rearp = (queue_node_t *)malloc(sizeof (queue_node_t));
11.             qp->frontp = qp->rearp;
12.       } else {                              /* adds to nonempty queue             */
13.             qp->rearp->restp =
14.                   (queue_node_t *)malloc(sizeof (queue_node_t));
15.             qp->rearp = qp->rearp->restp;
16.       }
17.       qp->rearp->element = ele;             /* defines newly added node           */
18.       qp->rearp->restp = NULL;
19.       ++(qp->size);
20. }
21.
```

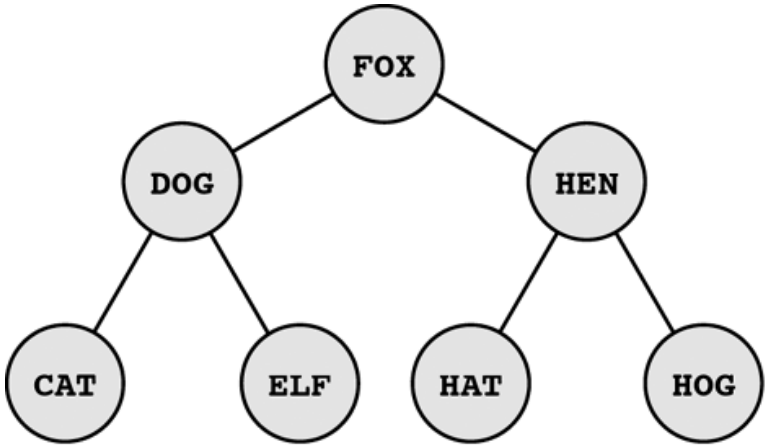

Before
Brown
E(conomy)
2
Watson
B(usiness)
1
Carson
F(irstClass)
2
q.frontp
q.rearp
q.size
3
remove_fr om_q(&q);


During
function
call
to_freep
Brown
E(conomy)
2
Watson
B(usiness)
1
Carson
F(irstClass)
2
q.frontp
q.rearp
q.size
2

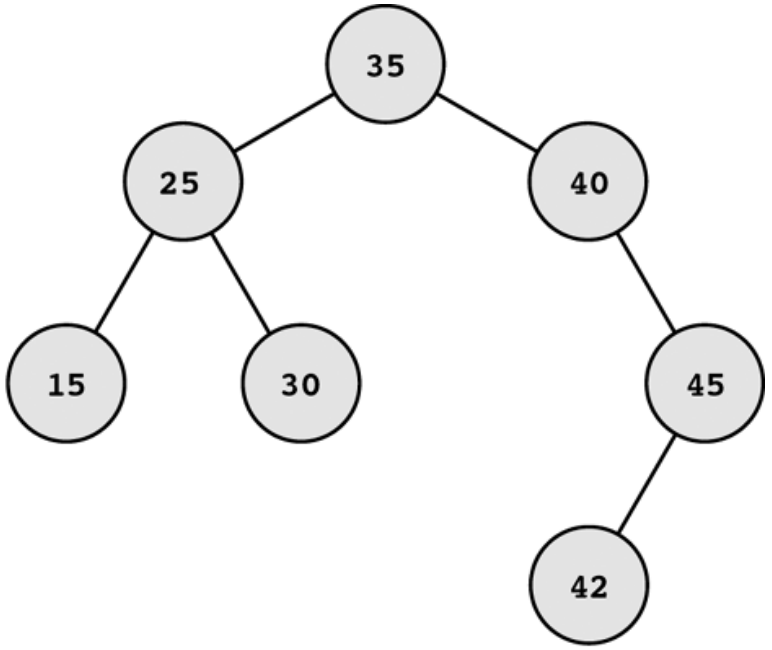

Value returned
by remove_from_q
Brown
E(conomy)
2

```
21.
22.  /*
23.   *  Removes and frees first node of queue, returning value stored there.
24.   *  Pre:  queue is not empty
25.   */
26.  queue_element_t
27.  remove_from_q(queue_t *qp) /* input/output - queue */
28.  {
30.       queue_node_t    *to_freep;    /* pointer to node removed                    */
31.       queue_element_t  ans;         /* initial queue value which is to
32.                                        be returned                                */
33.        to_freep = qp->frontp;           /* saves pointer to node being deleted */
34.        ans = to_freep->element;         /* retrieves value to return           */
35.        qp->frontp = to_freep->restp;    /* deletes first node                  */
36.        free(to_freep);                  /* deallocates space                   */
37.        --(qp->size);
38.
39.        if (qp->size == 0)               /* queue's ONLY node was deleted       */
40.              qp->rearp = NULL;
41.
42.        return (ans);
43.  }
```

Binary Trees



(a)

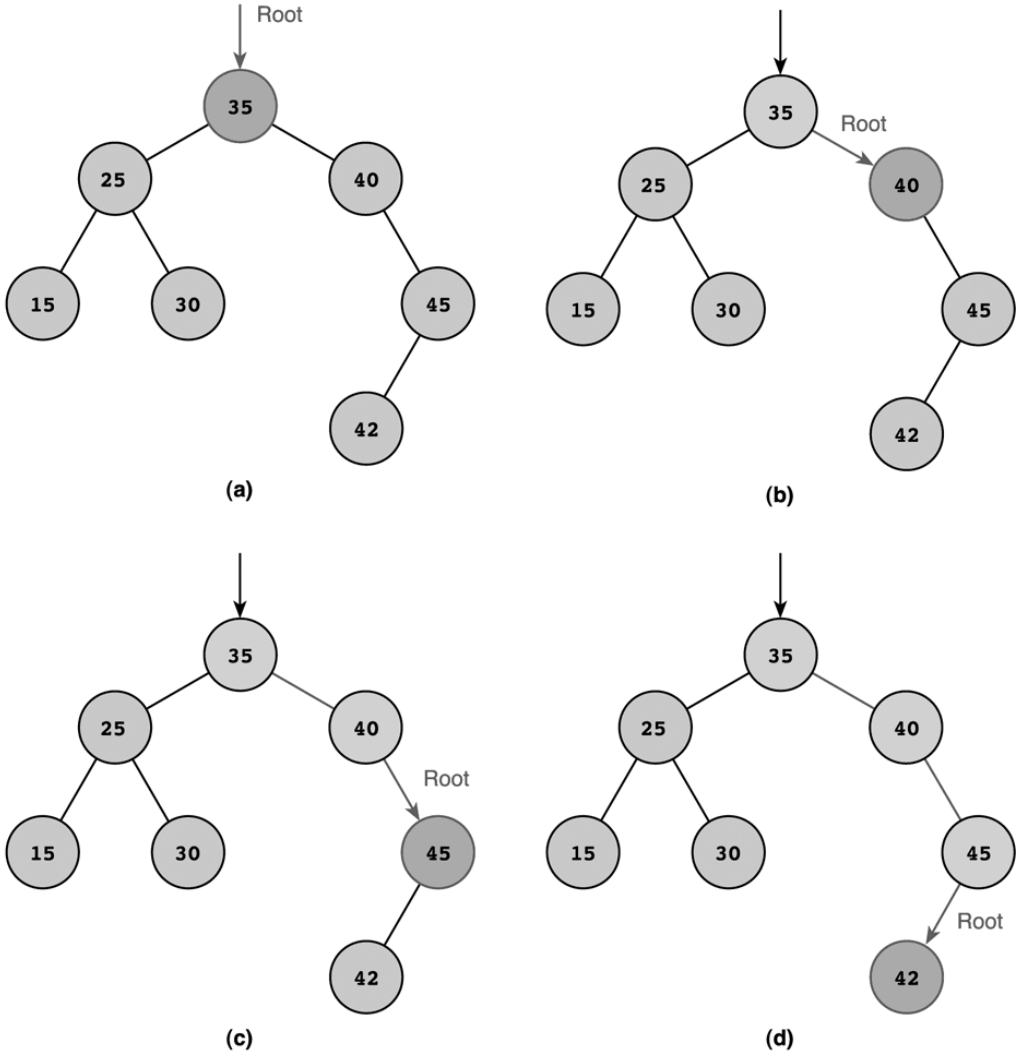


(b)

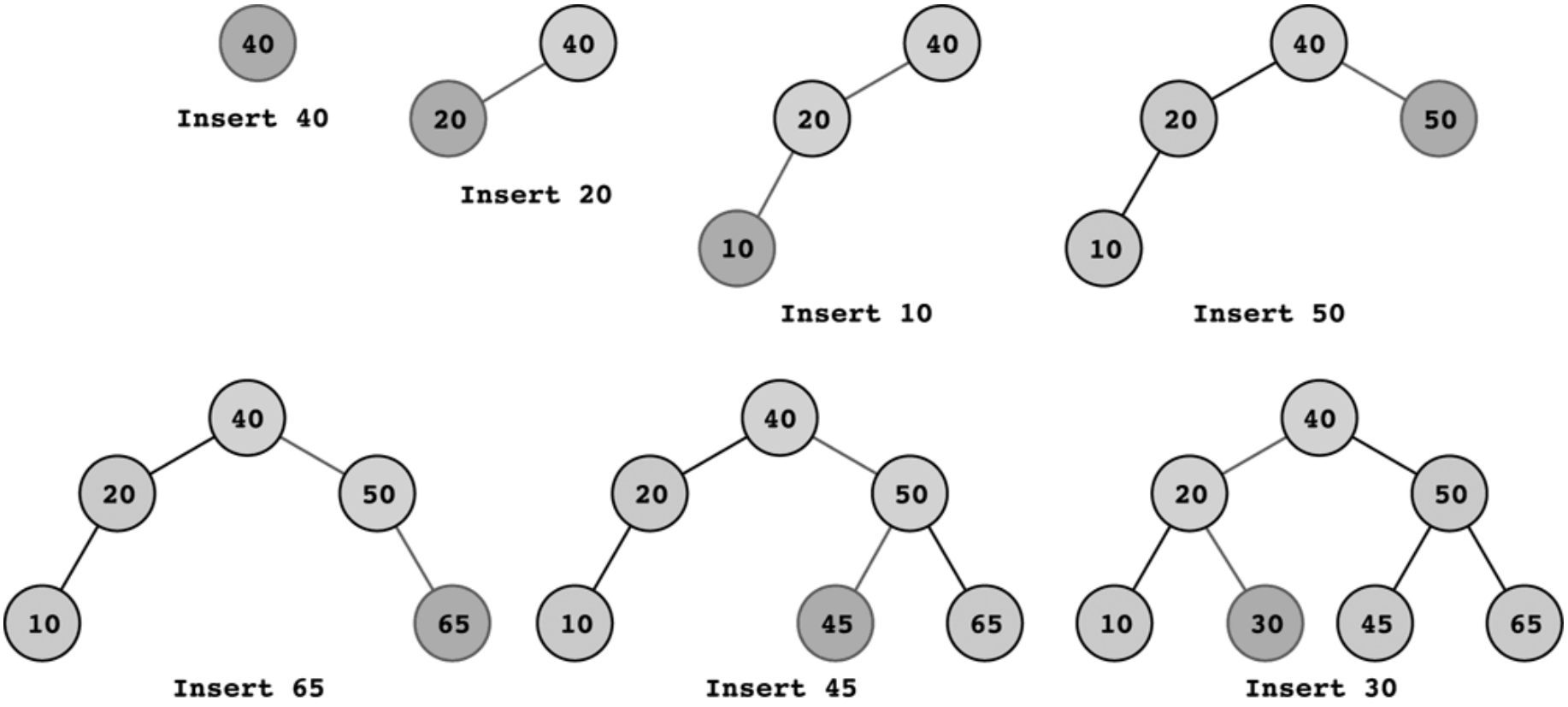
Binary Search Trees

- How to implement?

Binary Tree Search for 42



Building a Binary Search Tree



Creating a Binary Search Tree

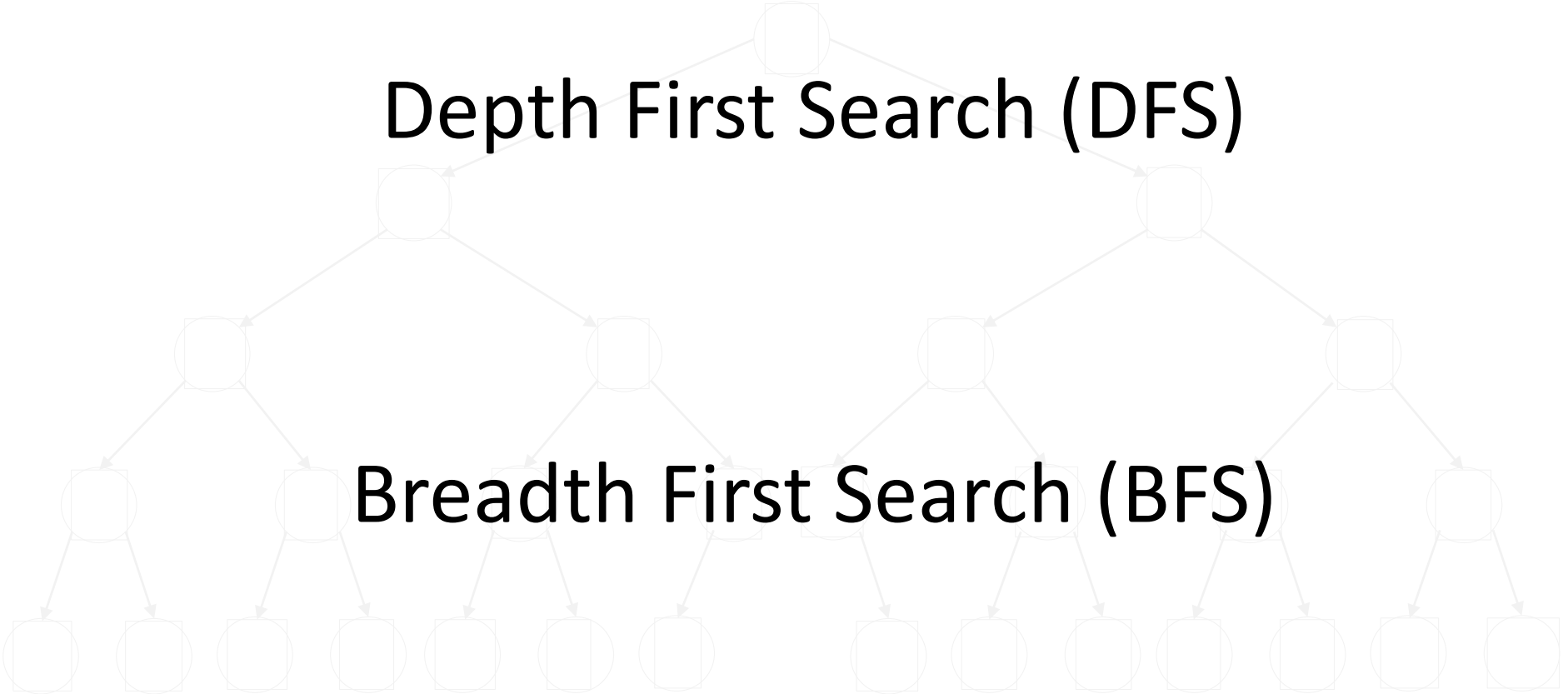
```
1.  /*
2.   *  Create and display a binary search tree of integer keys.
3.   */
4.
5.  #include <stdio.h>
6.  #include <stdlib.h>
7.
8.  #define TYPED_ALLOC(type) (type *)malloc(sizeof (type))
9.
10. typedef struct tree_node_s {
11.       int                   key;
12.       struct tree_node_s *leftp, *rightp;
13. } tree_node_t;
14.
15. tree_node_t *tree_insert(tree_node_t *rootp, int new_key);
16. void tree_inorder(tree_node_t *rootp);
17.
```

```
17.
18.  int
19.  main(void)
20.  {
21.        tree_node_t *bs_treep;    /* binary search tree              */
22.        int          data_key;    /* input - keys for tree           */
23.        int          status;      /* status of input operation       */
24.
25.        bs_treep = NULL;    /* Initially, tree is empty */
26.
27.        /* As long as valid data remains, scan and insert keys,
28.           displaying tree after each insertion. */
29.        for  (status = scanf("%d", &data_key);
30.              status == 1;
31.              status = scanf("%d", &data_key)) {
32.            bs_treep = tree_insert(bs_treep, data_key);
33.            printf("Tree after insertion of %d:\n", data_key);
34.            tree_inorder(bs_treep);
35.        }
36.
37.        if (status == 0) {
38.              printf("Invalid data >>%c\n", getchar());
39.        } else {
40.              printf("Final binary search tree:\n");
41.              tree_inorder(bs_treep);
42.        }
43.
44.        return (0);
45.  }
46.
```


(continued)

```
47. /*
48.  *  Insert a new key in a binary search tree.  If key is a duplicate,
49.  *  there is no insertion.
50.  *  Pre:  rootp points to the root node of a binary search tree
51.  *  Post: Tree returned includes new key and retains binary
52.  *        search tree properties.
53.  */
54. tree_node_t *
55. tree_insert(tree_node_t *rootp,   /* input/output - root node of
56.                                     binary search tree    */
57.            int          new_key) /* input - key to insert */
58. {
59.       if (rootp == NULL) {              /* Simple Case 1 - Empty tree   */
60.             rootp = TYPED_ALLOC(tree_node_t);
61.             rootp->key = new_key;
62.             rootp->leftp = NULL;
63.             rootp->rightp = NULL;
64.       } else if (new_key == rootp->key) {            /* Simple Case 2 */
65.             /* duplicate key - no insertion                          */
66.       } else if (new_key < rootp->key) {             /* Insert in     */
67.             rootp->leftp = tree_insert               /* left subtree  */
68.                            (rootp->leftp, new_key);
69.       } else {                        /* Insert in right subtree */
70.             rootp->rightp = tree_insert(rootp->rightp,
71.                                         new_key);
72.       }
73.
74.       return (rootp);
75. }
```

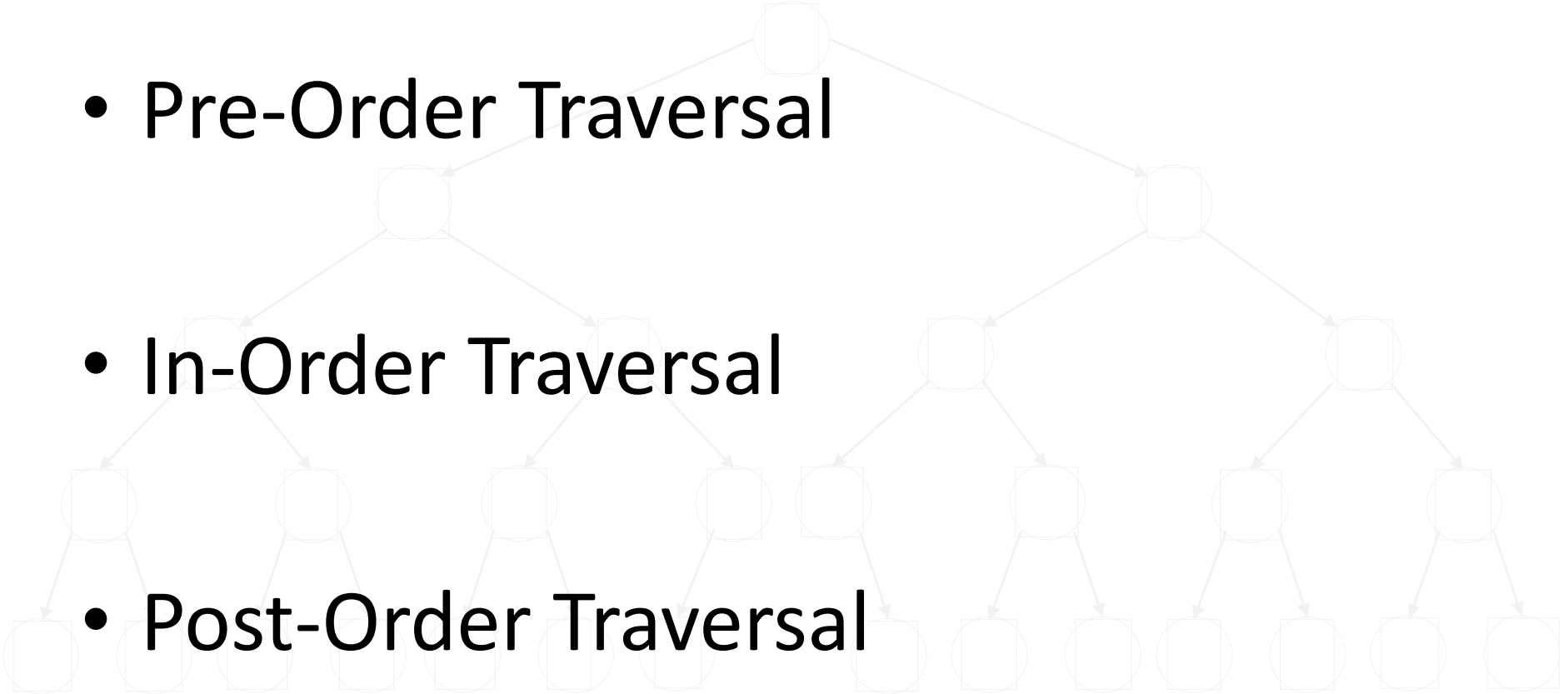
Binary Tree Traversal



Depth First Search (DFS)

Breadth First Search (BFS)

Binary Tree Traversal (DFS)



- Pre-Order Traversal
- In-Order Traversal
- Post-Order Traversal

Binary Tree Traversal

Pre-Order Traversal

```
preorder(node)
    If (node == null)
        return;
    visit(node)
    preorder(node.left)
    preorder(node.right)
```

Binary Tree Traversal

In-Order Traversal

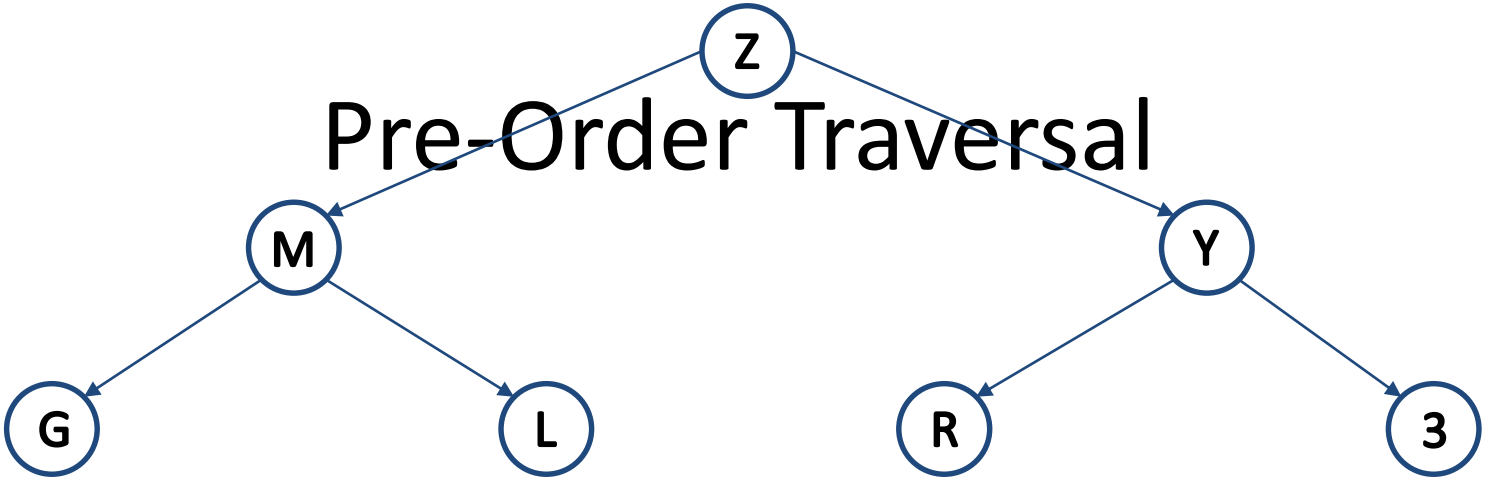
```
inorder(node)
    If (node == null)
        return;
    inorder(node.left)
    visit(node)
    inorder(node.right)
```

Binary Tree Traversal

Post-Order Traversal

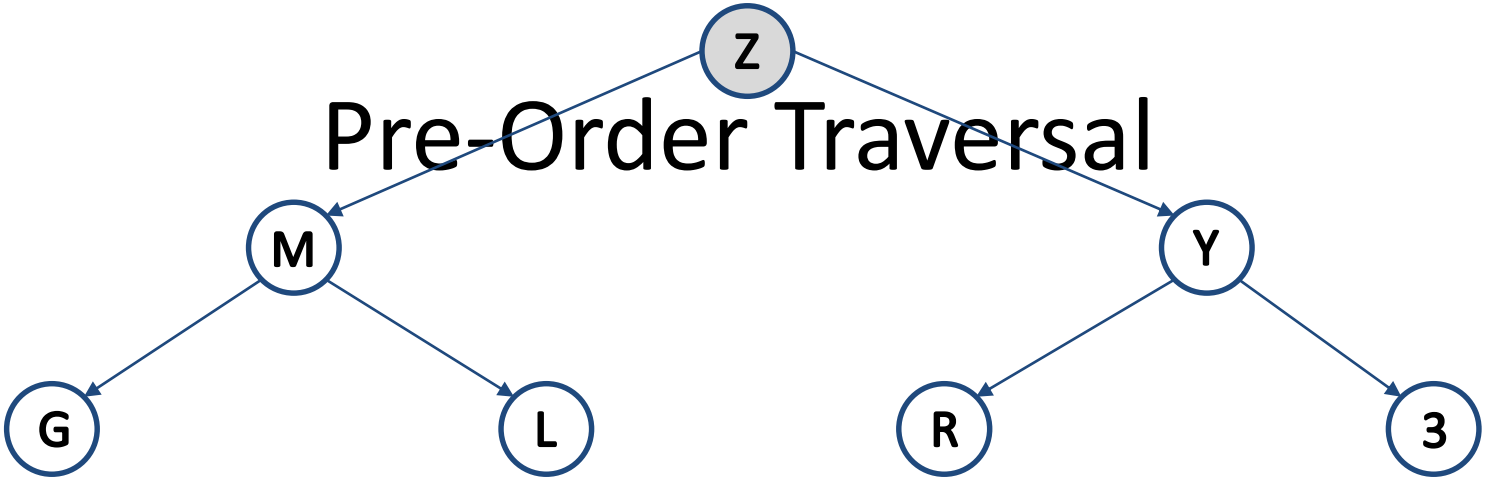
```
postorder(node)
    If (node == null)
        return;
    postorder(node.left)
    postorder(node.right)
    visit(node)
```

Binary Tree Traversal



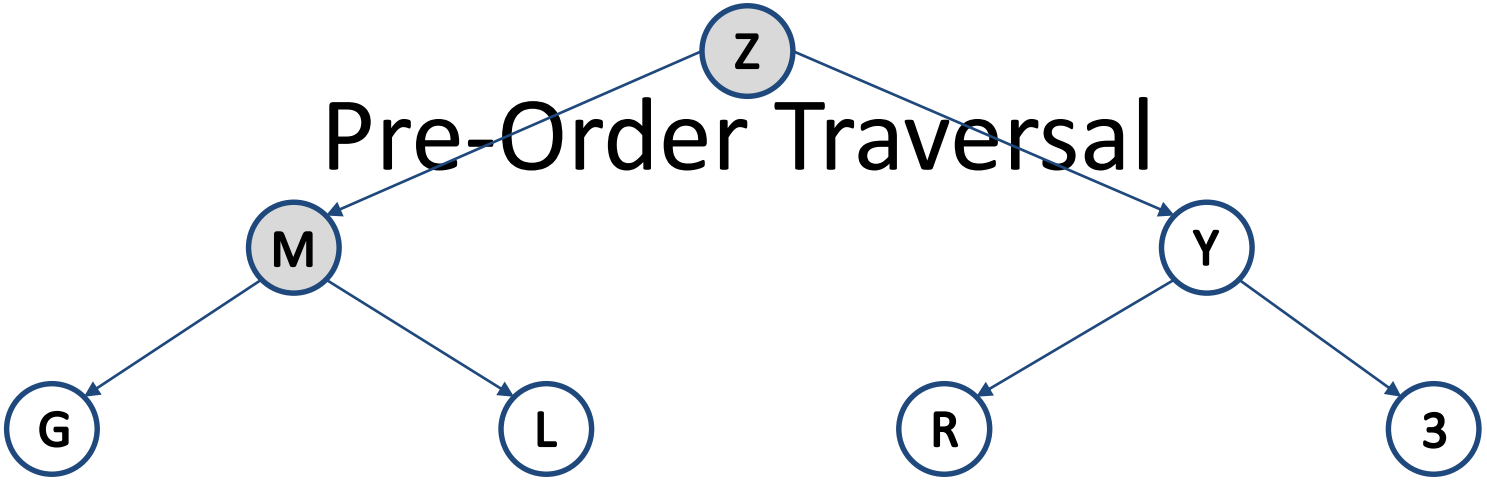
Pre-Order Traversal

Binary Tree Traversal



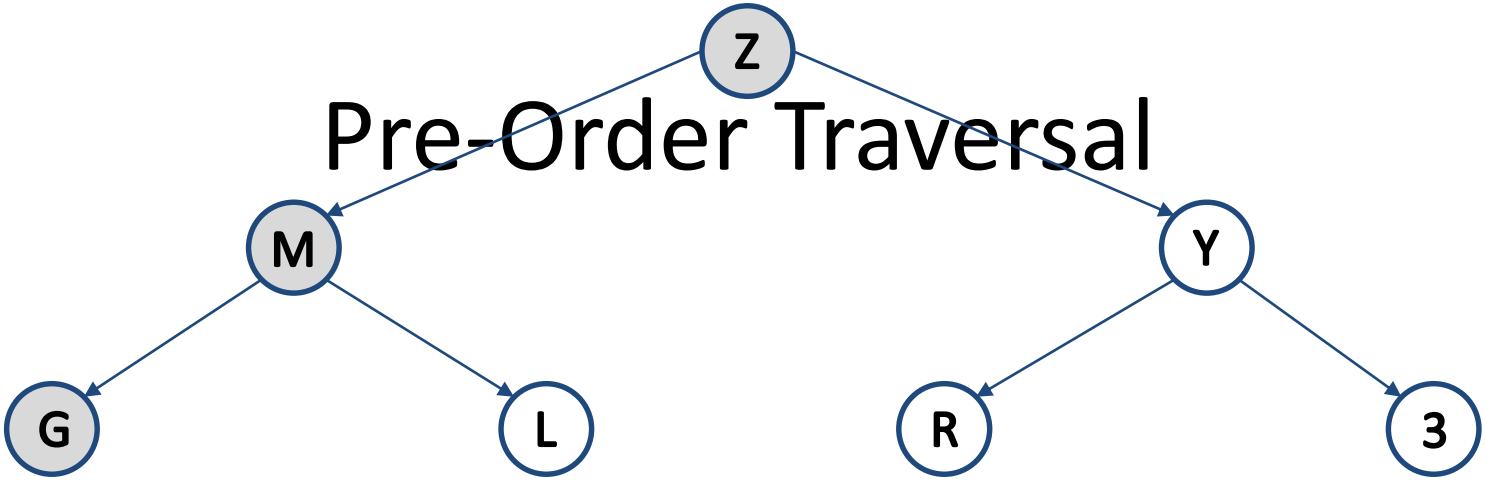
Pre-Order Traversal

Binary Tree Traversal



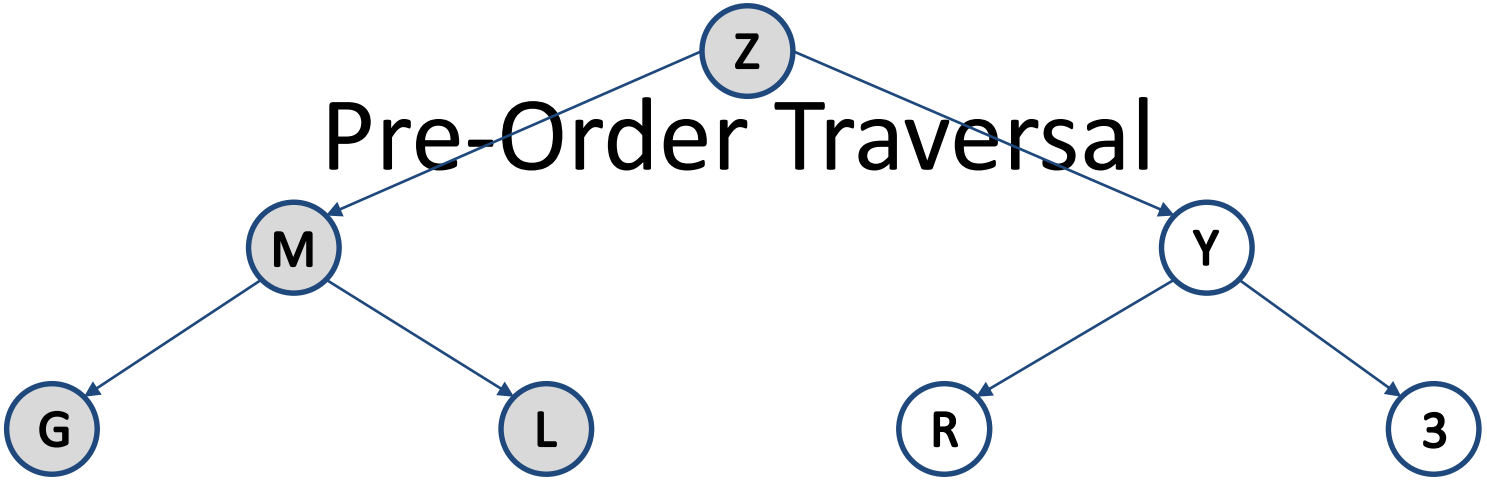
Pre-Order Traversal

Binary Tree Traversal



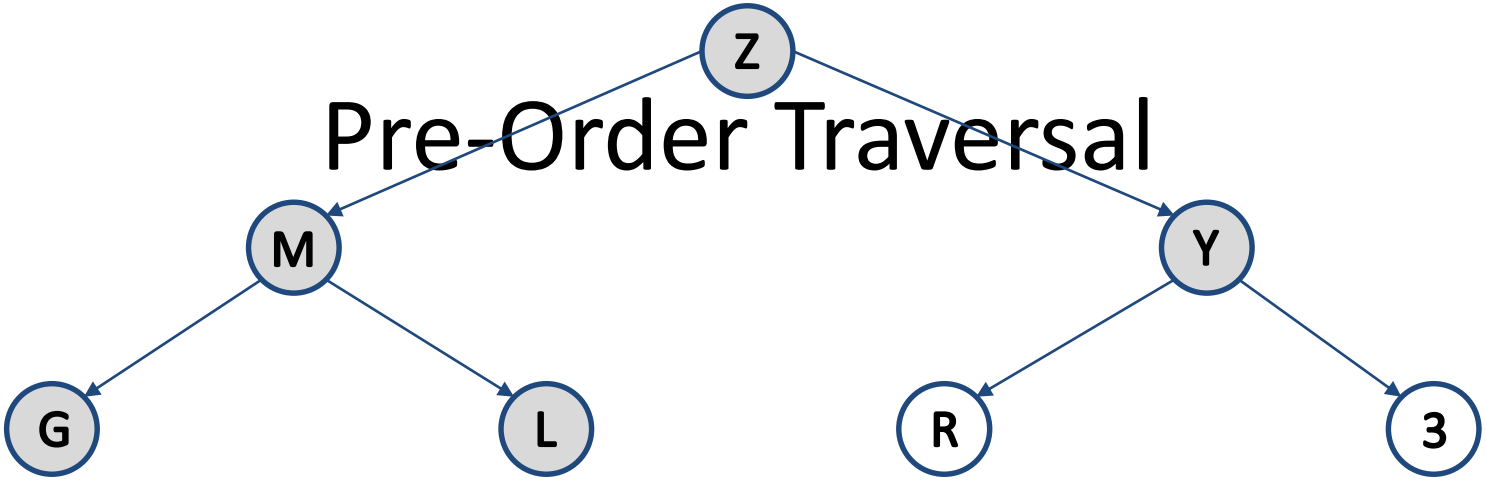
Pre-Order Traversal

Binary Tree Traversal



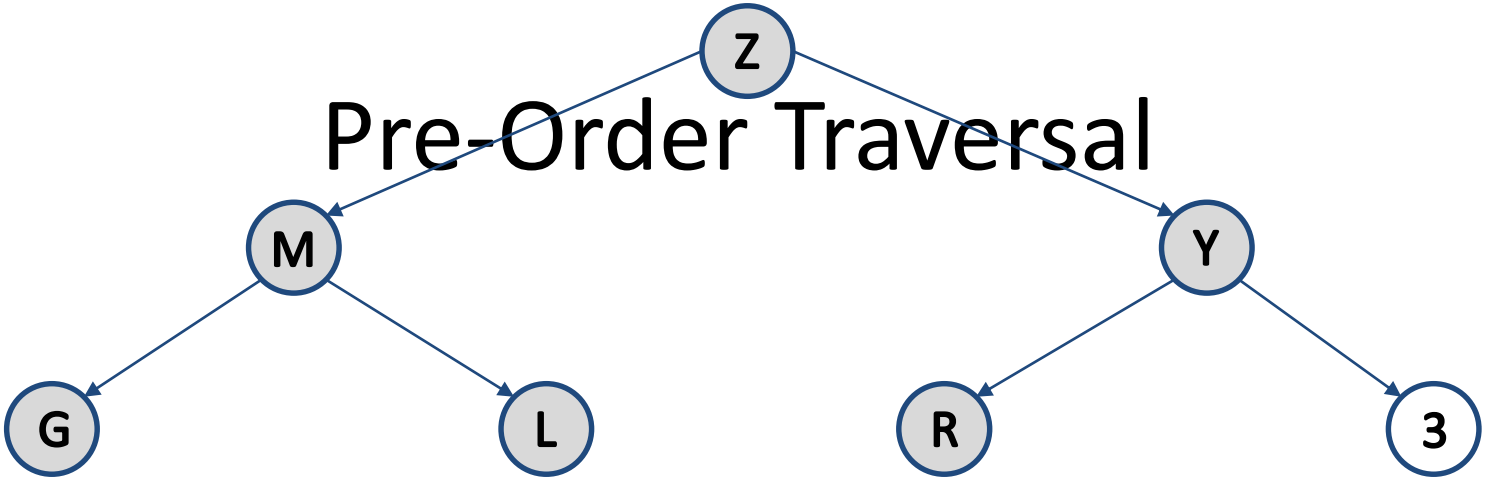
Pre-Order Traversal

Binary Tree Traversal



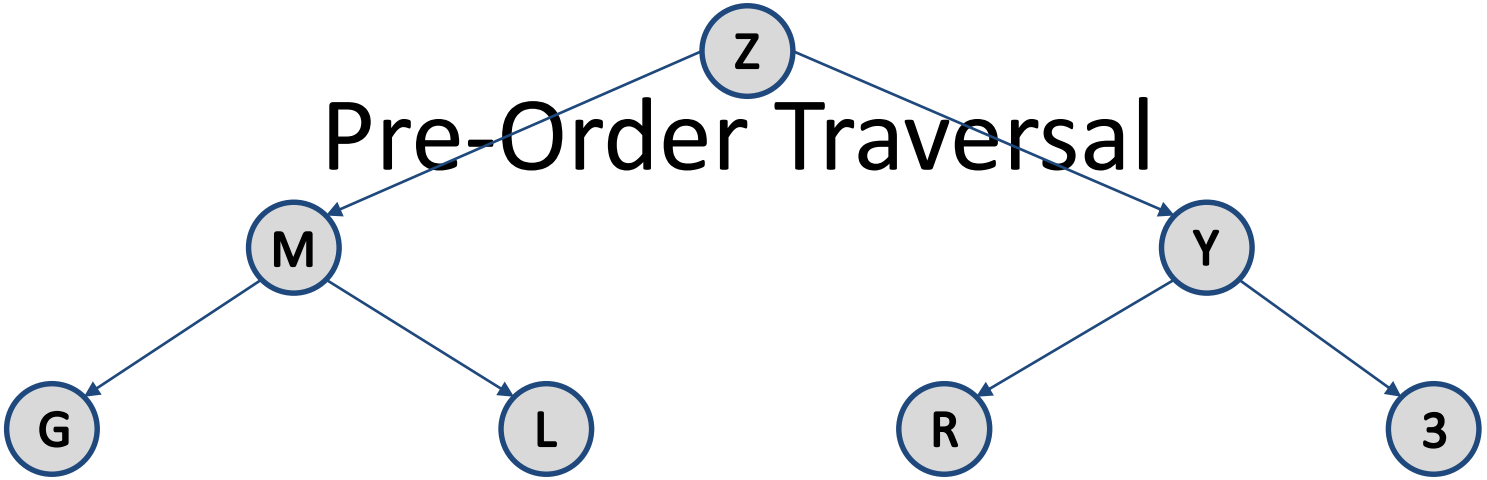
Pre-Order Traversal

Binary Tree Traversal



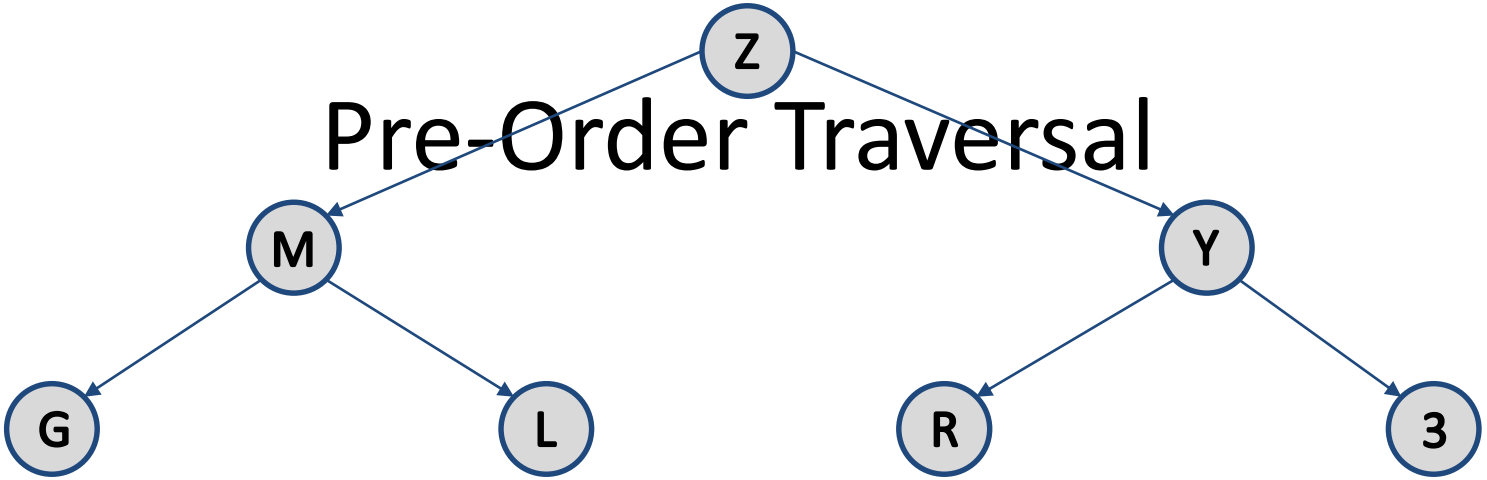
Pre-Order Traversal

Binary Tree Traversal



Pre-Order Traversal

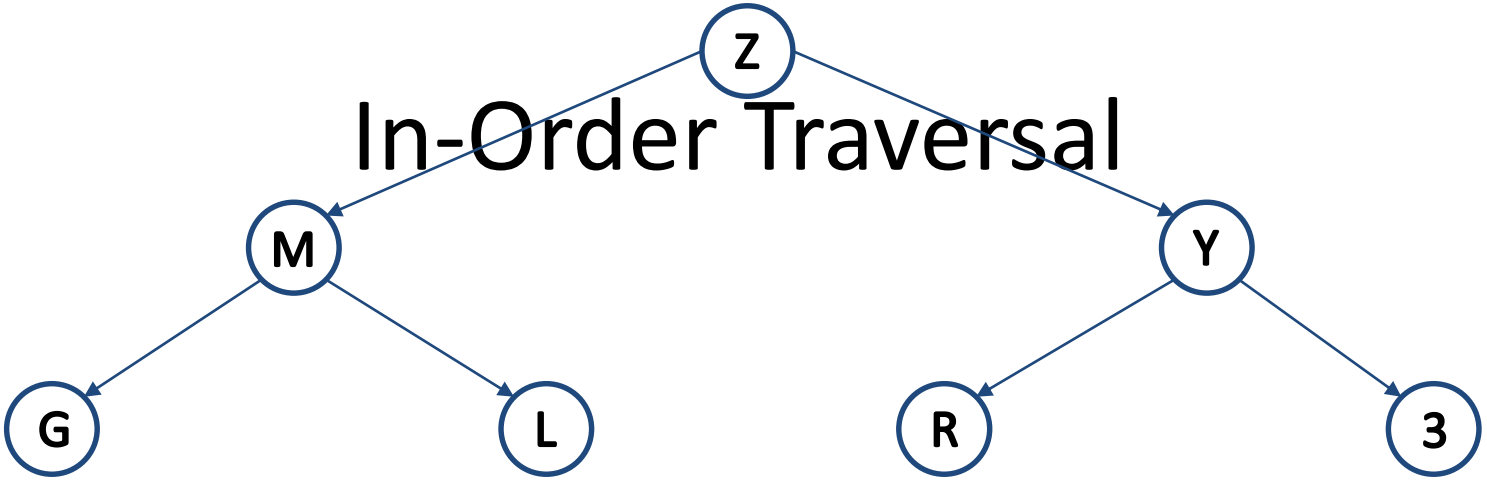
Binary Tree Traversal



Pre-Order Traversal

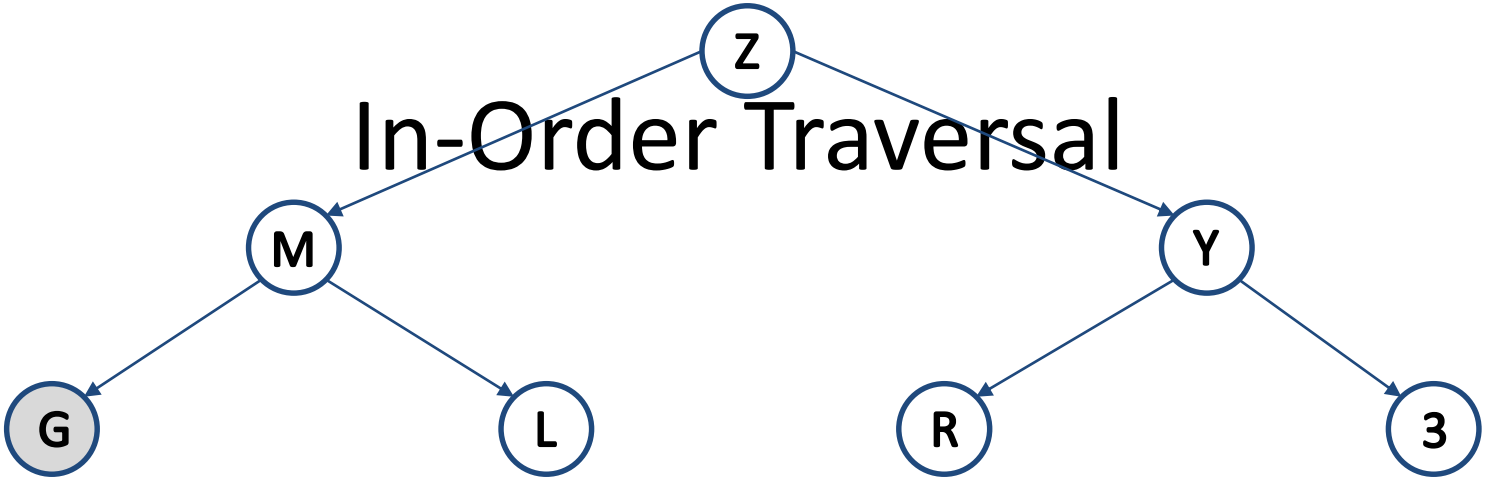
Z M G L Y R 3

Binary Tree Traversal

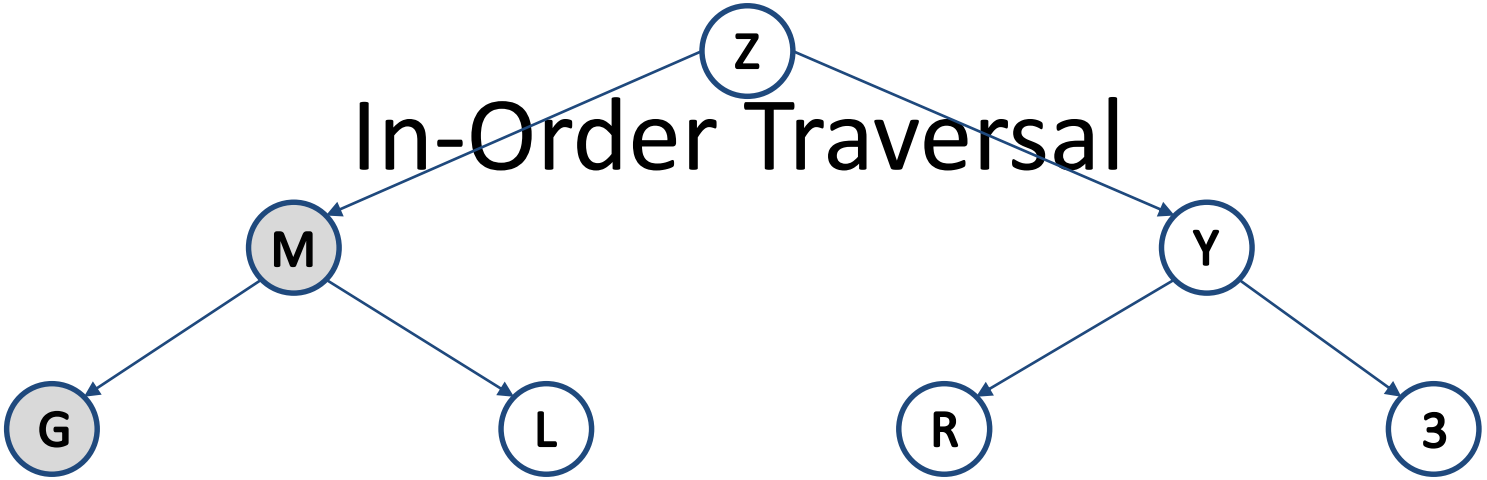


In-Order Traversal

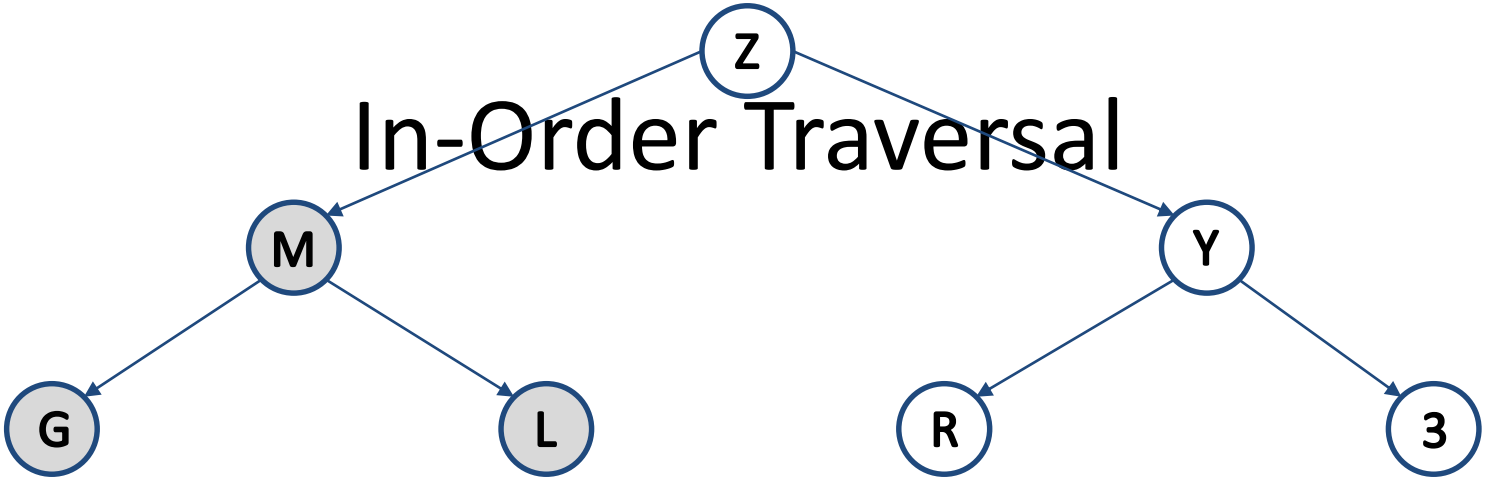
Binary Tree Traversal



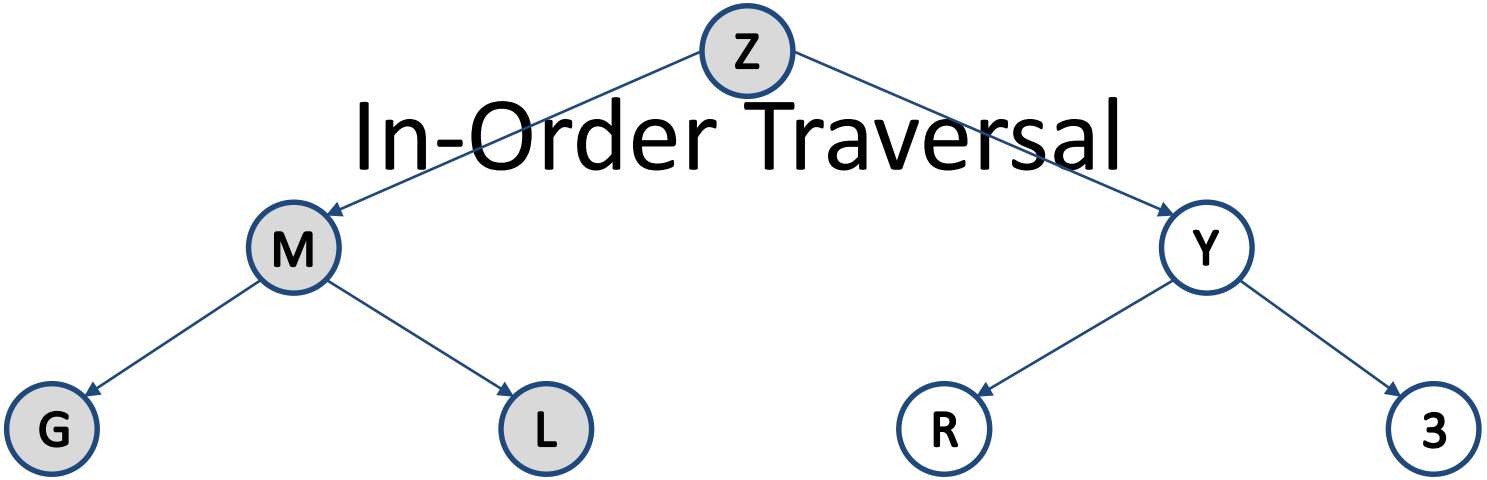
Binary Tree Traversal



Binary Tree Traversal

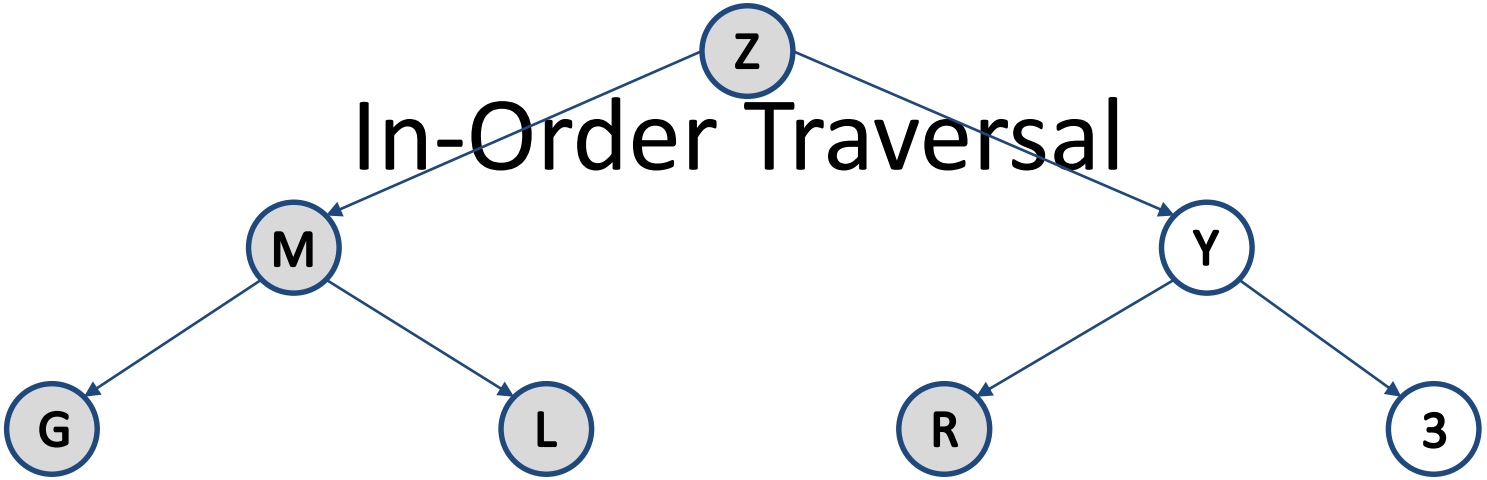


Binary Tree Traversal

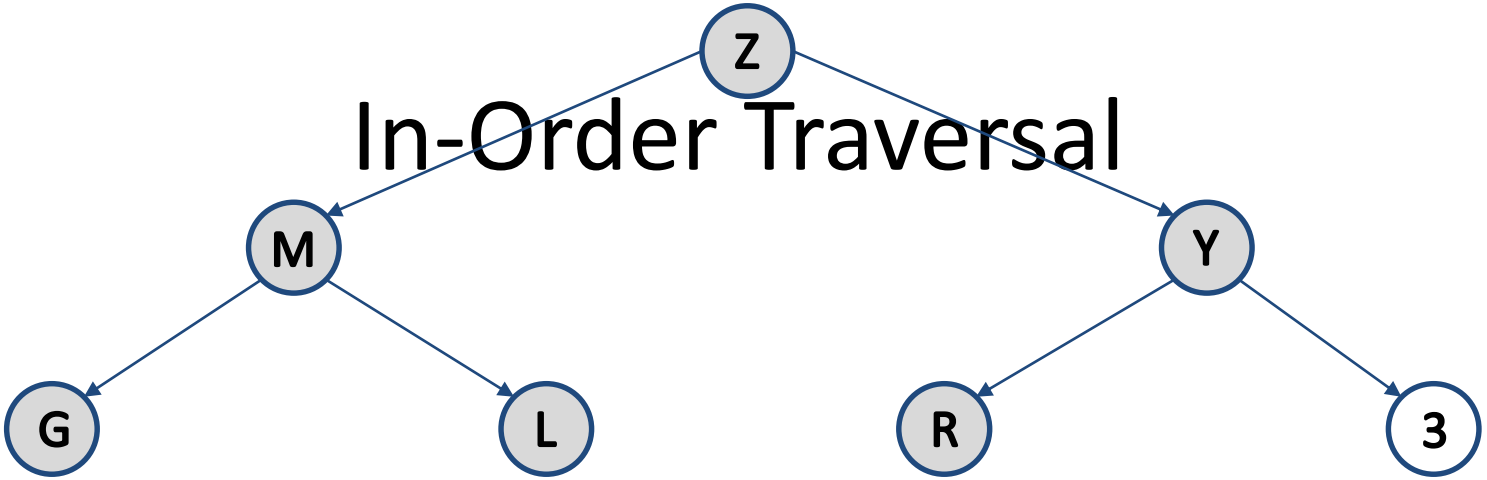


In-Order Traversal

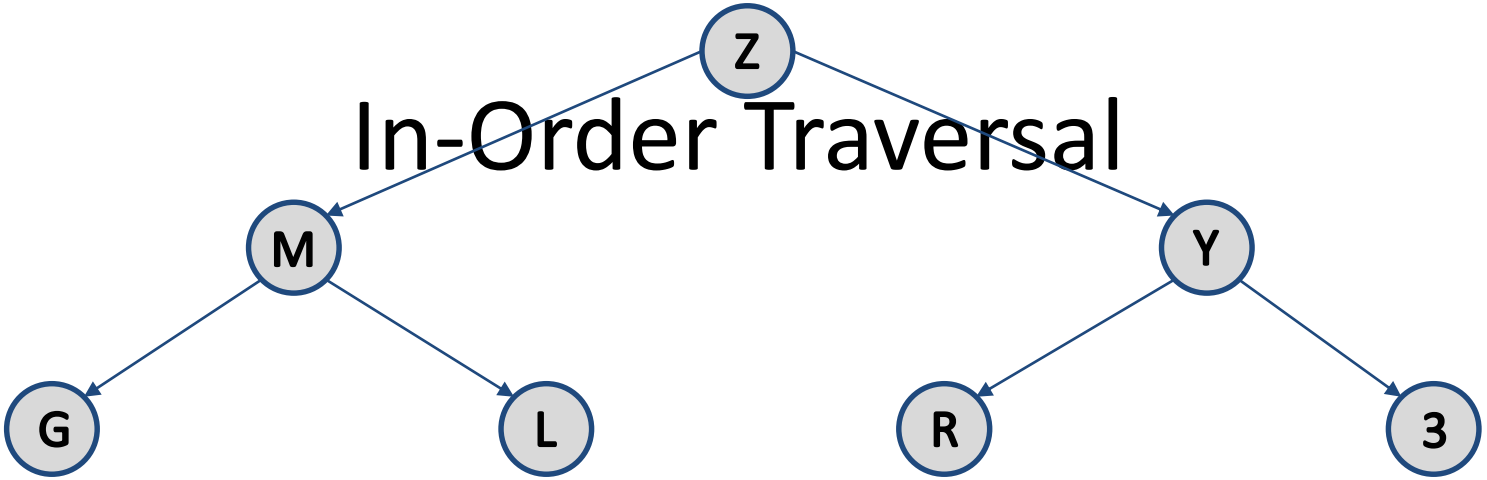
Binary Tree Traversal



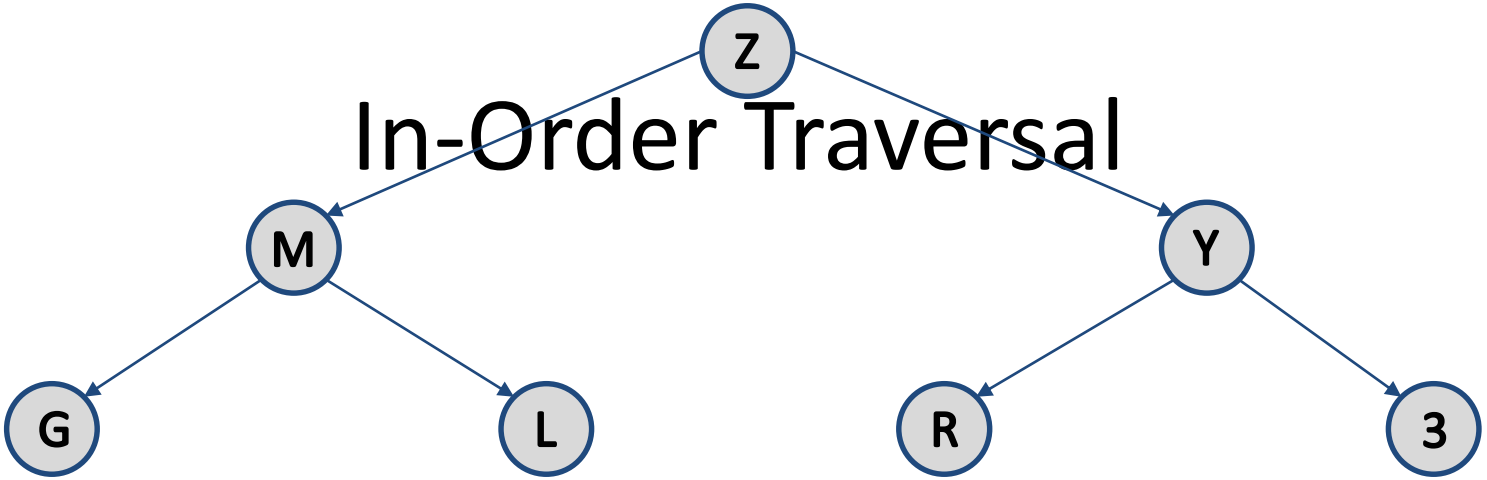
Binary Tree Traversal



Binary Tree Traversal

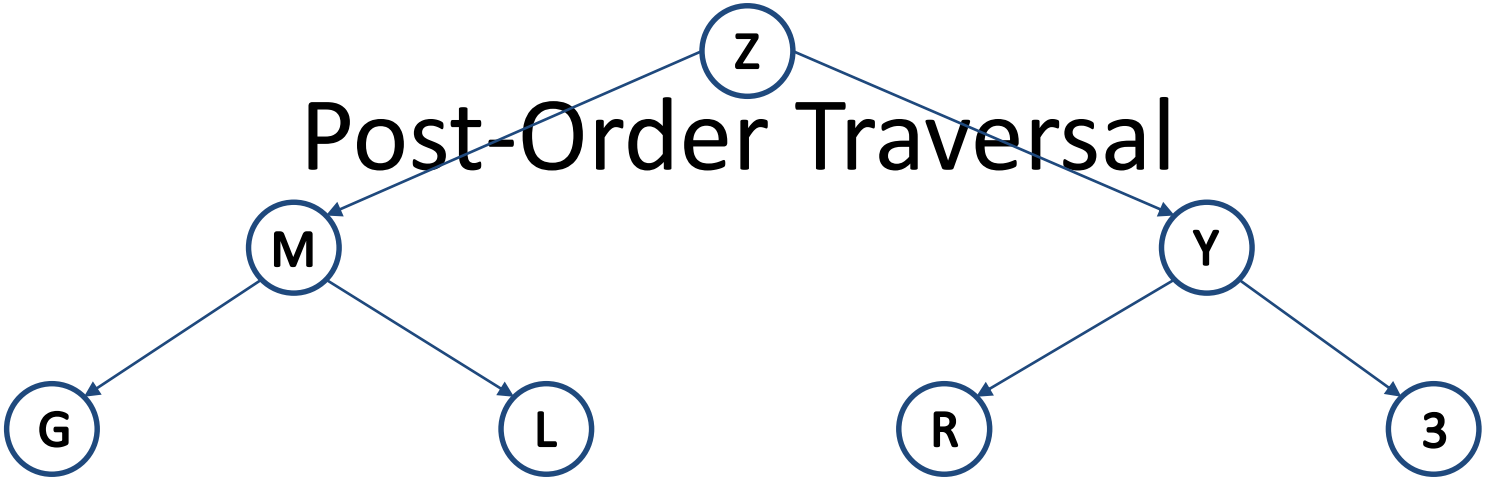


Binary Tree Traversal



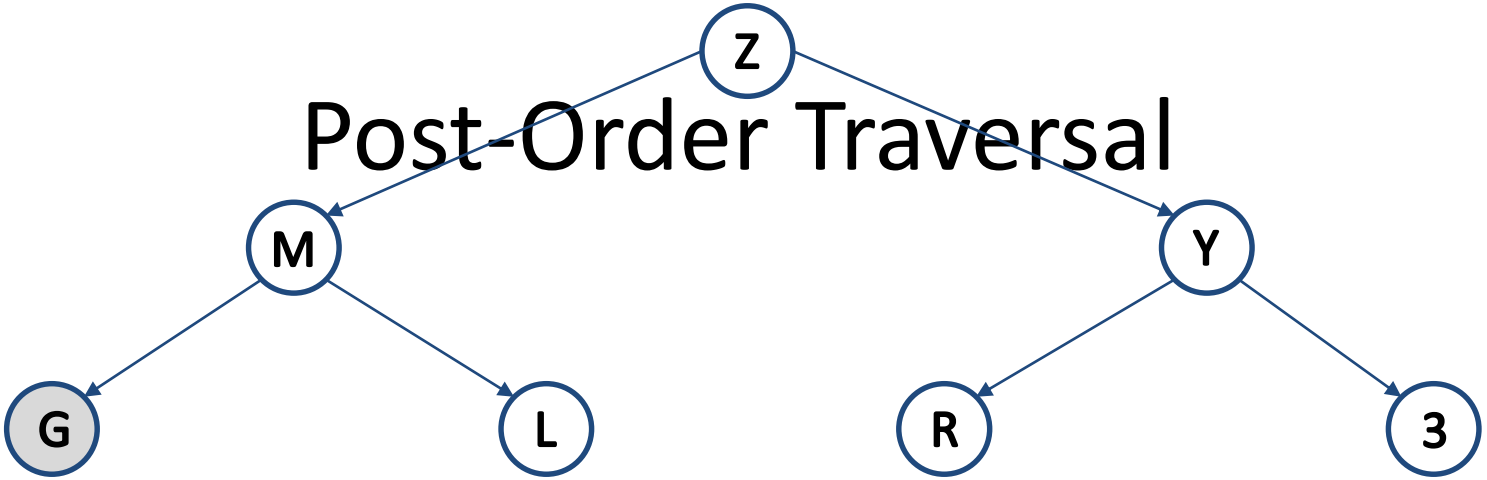
G M L Z Y R 3

Binary Tree Traversal



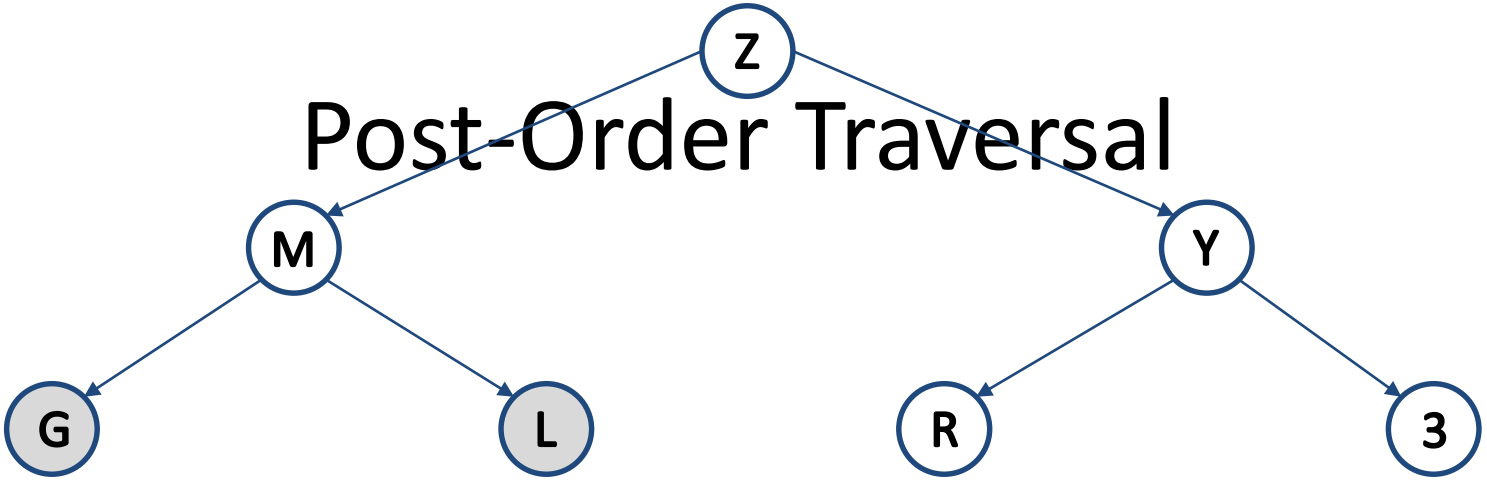
Post-Order Traversal

Binary Tree Traversal



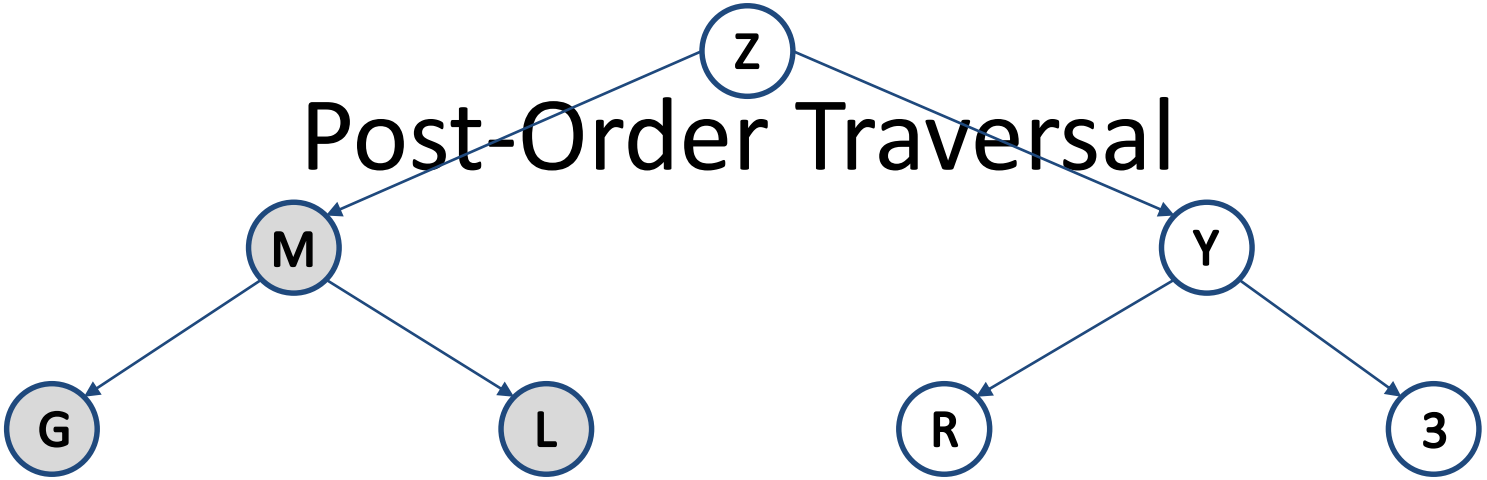
Post-Order Traversal

Binary Tree Traversal



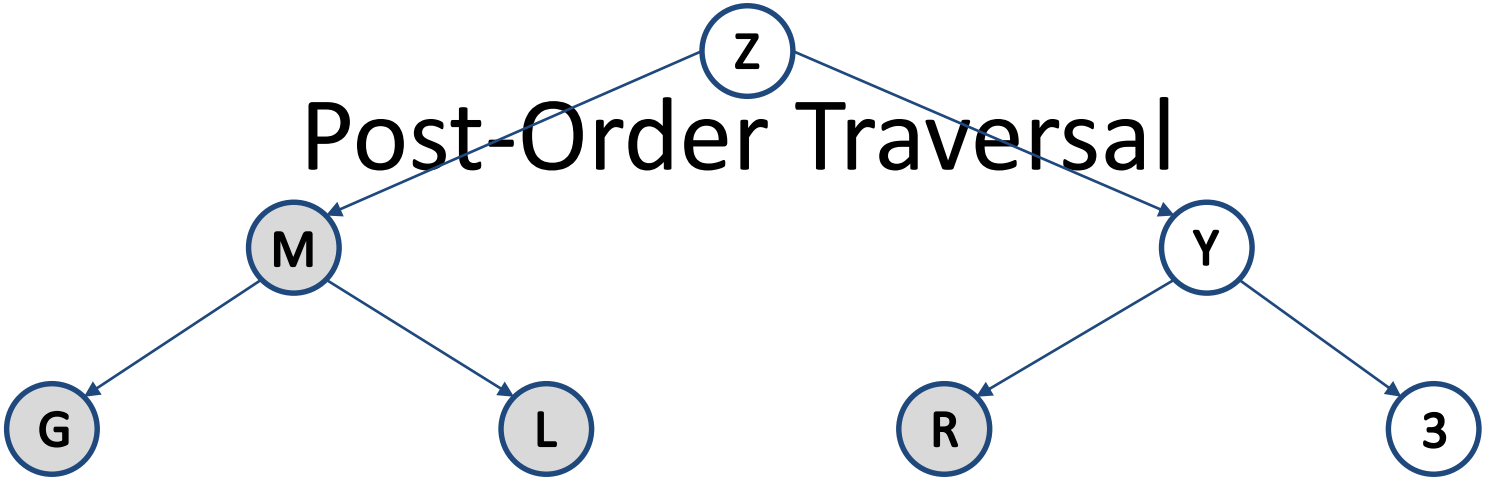
Post-Order Traversal

Binary Tree Traversal



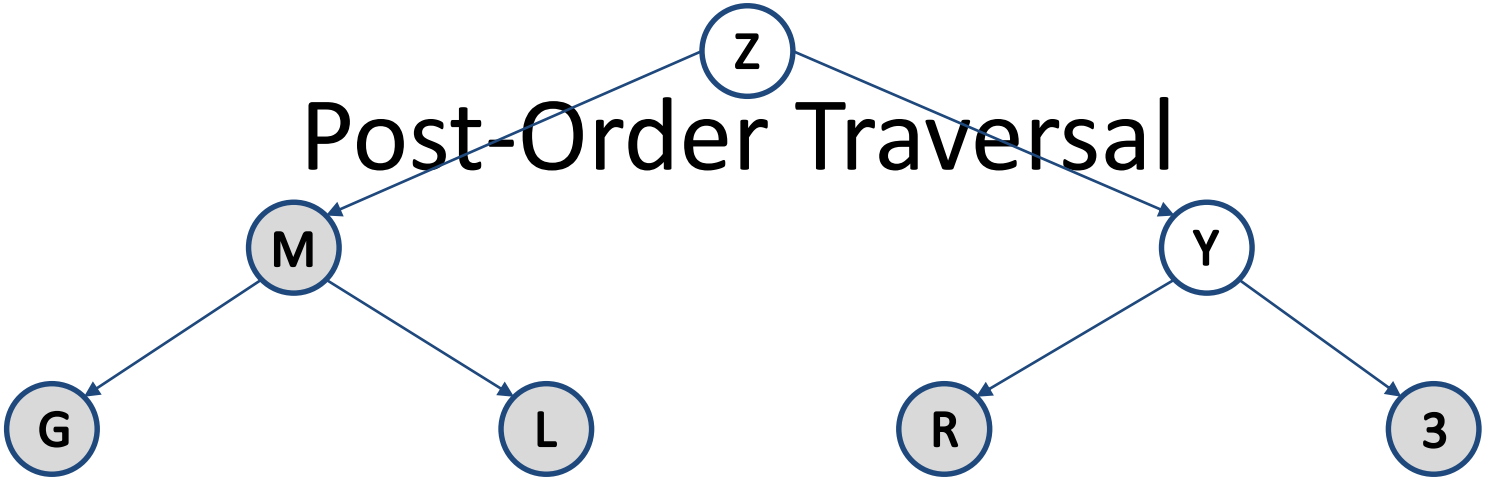
Post-Order Traversal

Binary Tree Traversal



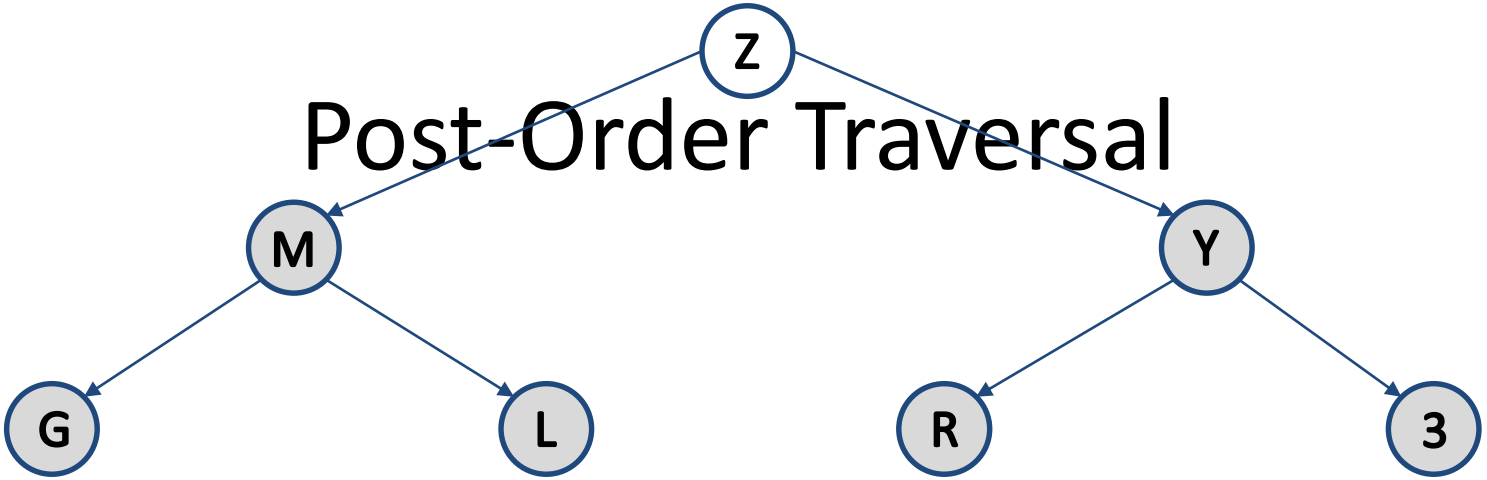
Post-Order Traversal

Binary Tree Traversal



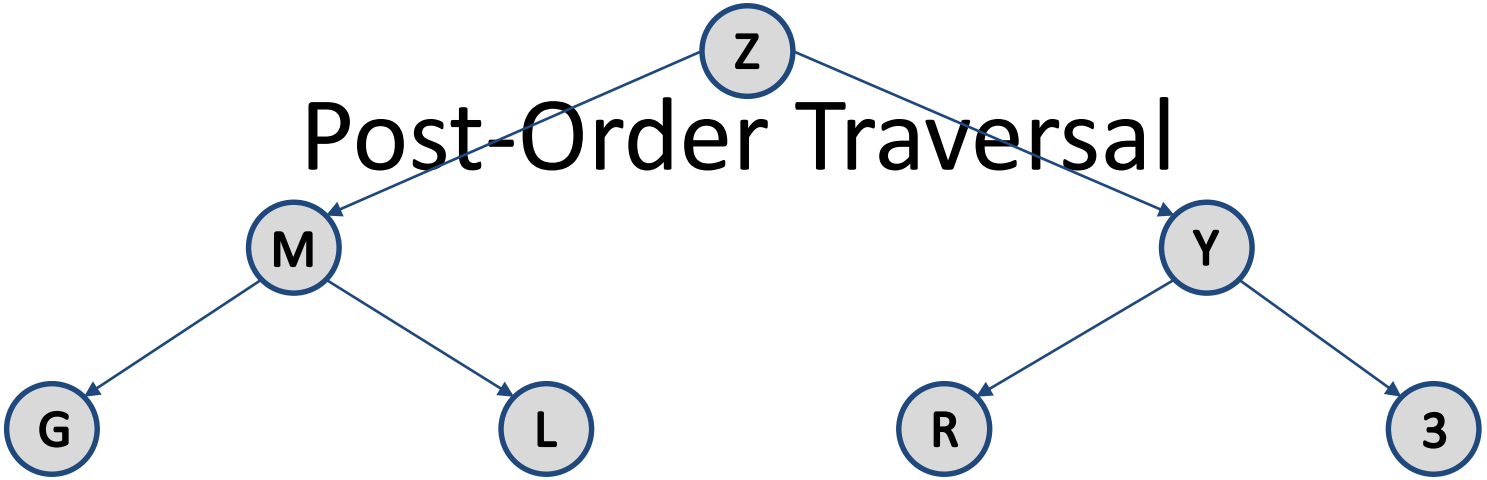
Post-Order Traversal

Binary Tree Traversal



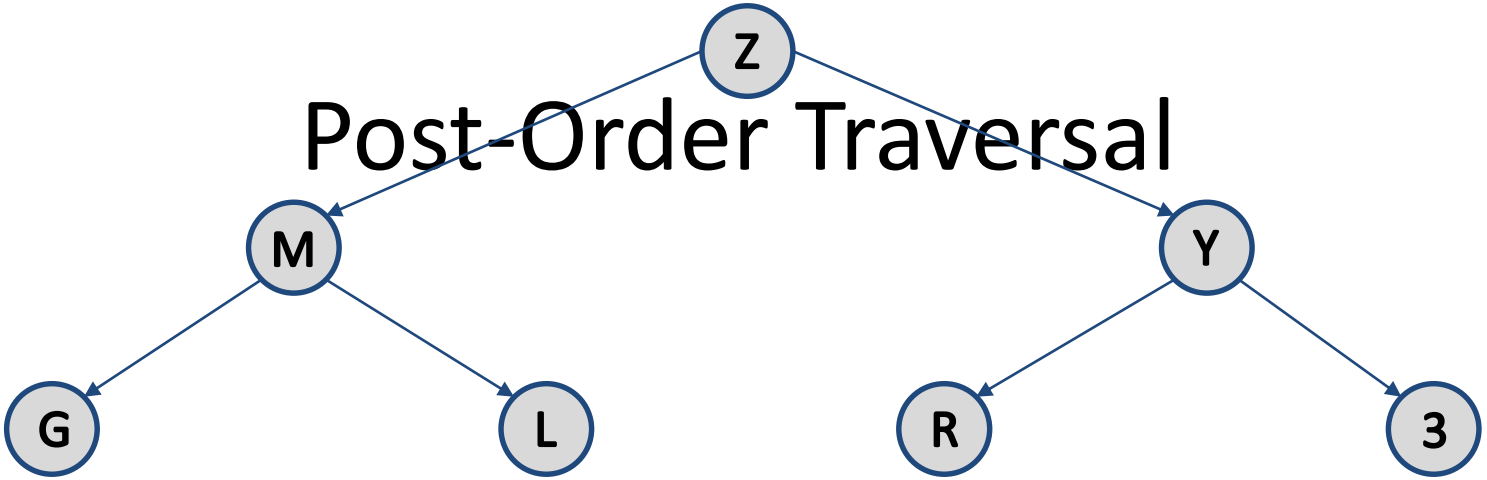
Post-Order Traversal

Binary Tree Traversal



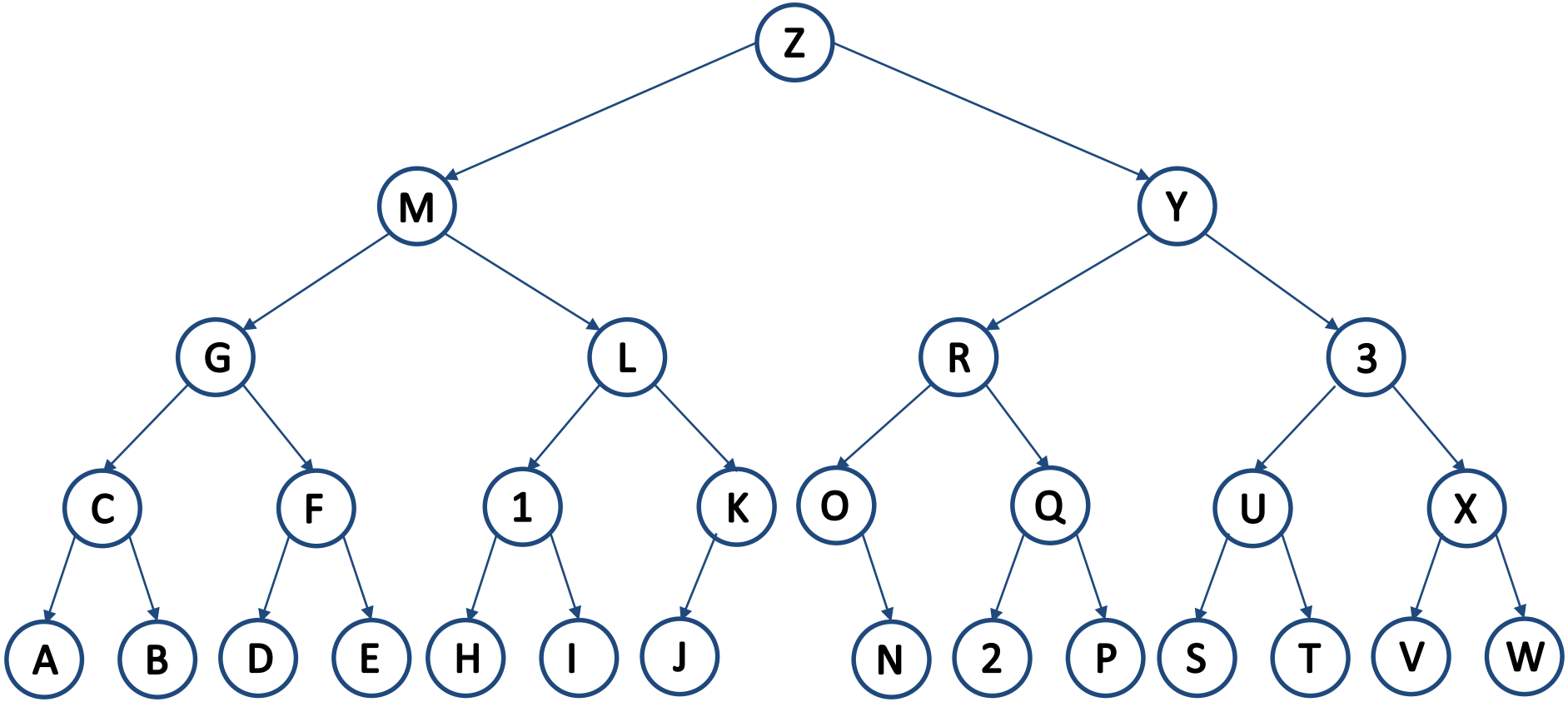
Post-Order Traversal

Binary Tree Traversal

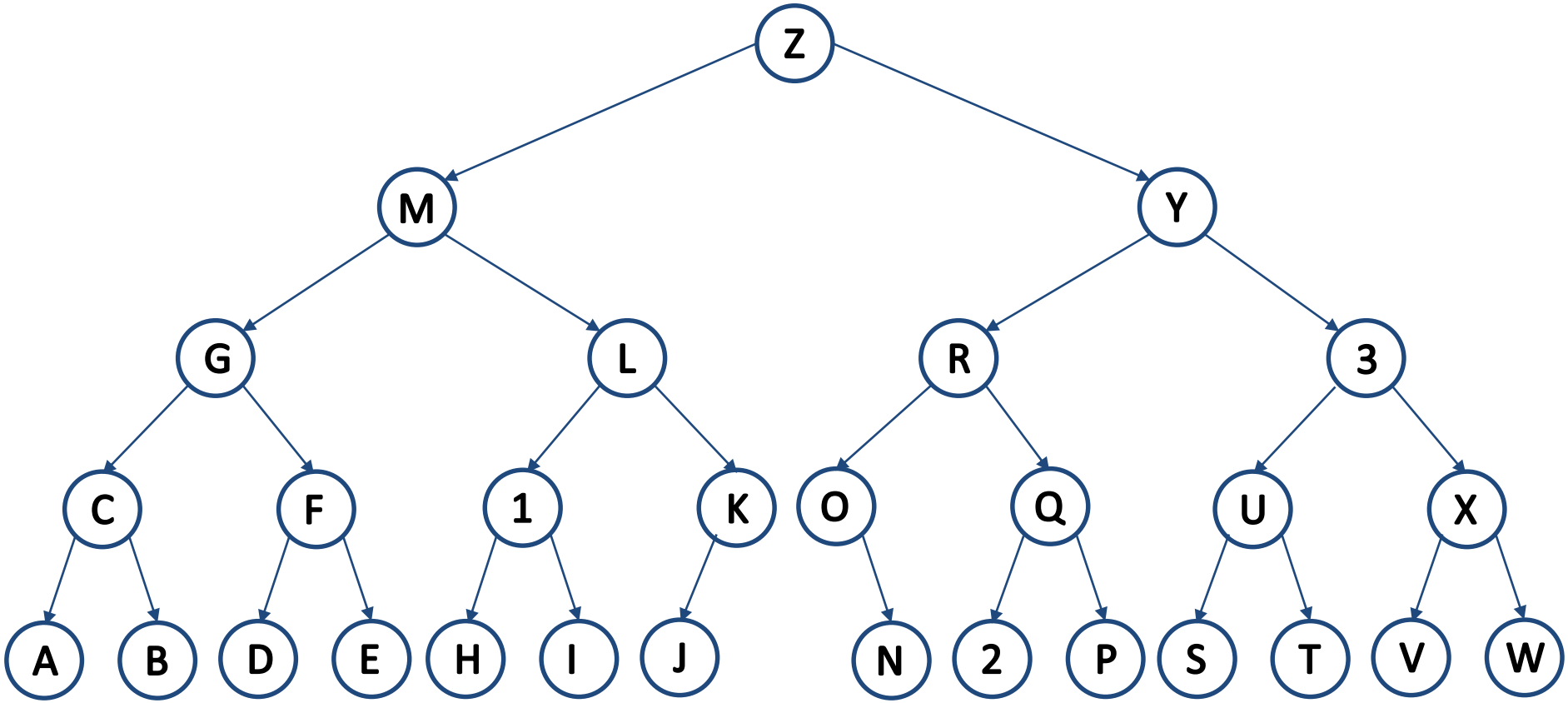


G L M R 3 Y Z

Binary Tree Pre-Oder Traversal

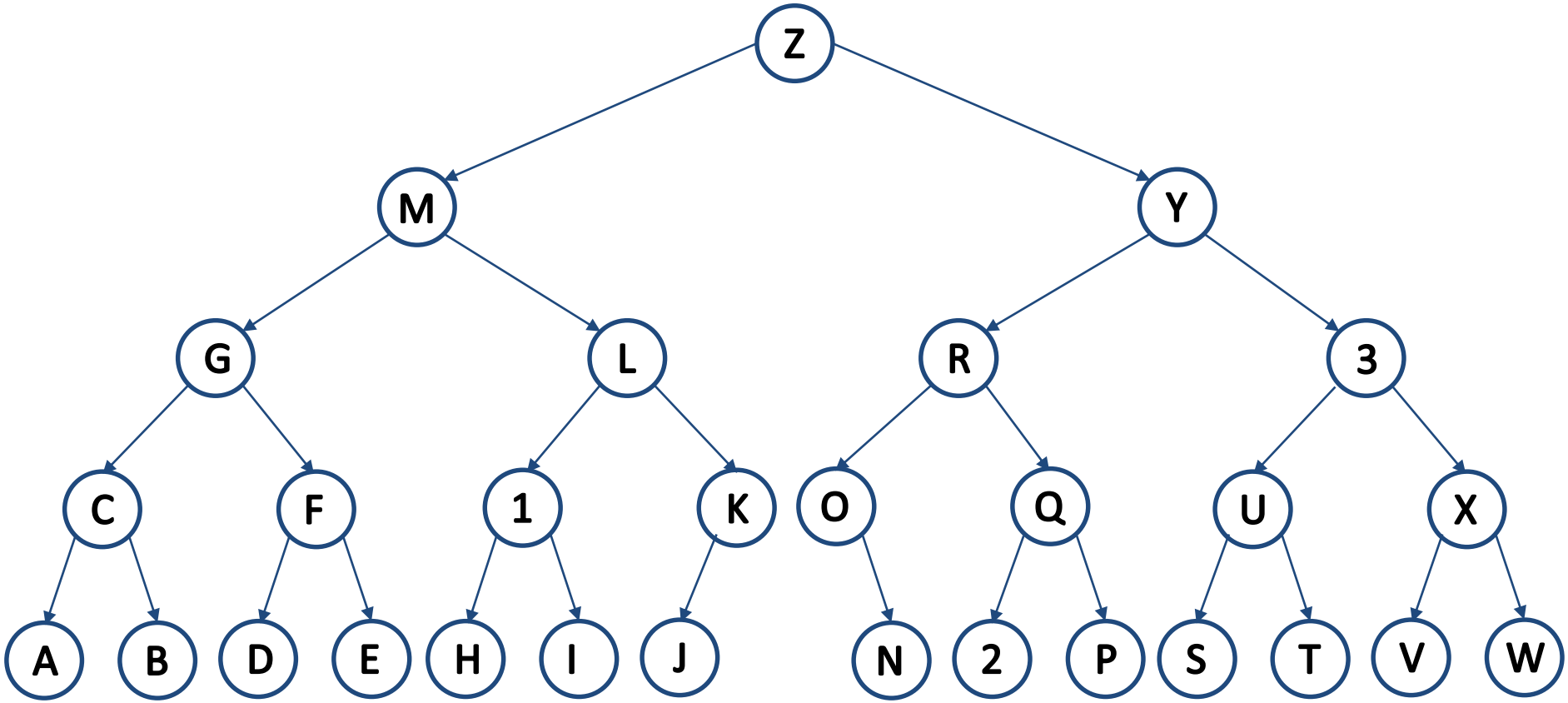


Binary Tree Pre-Oder Traversal

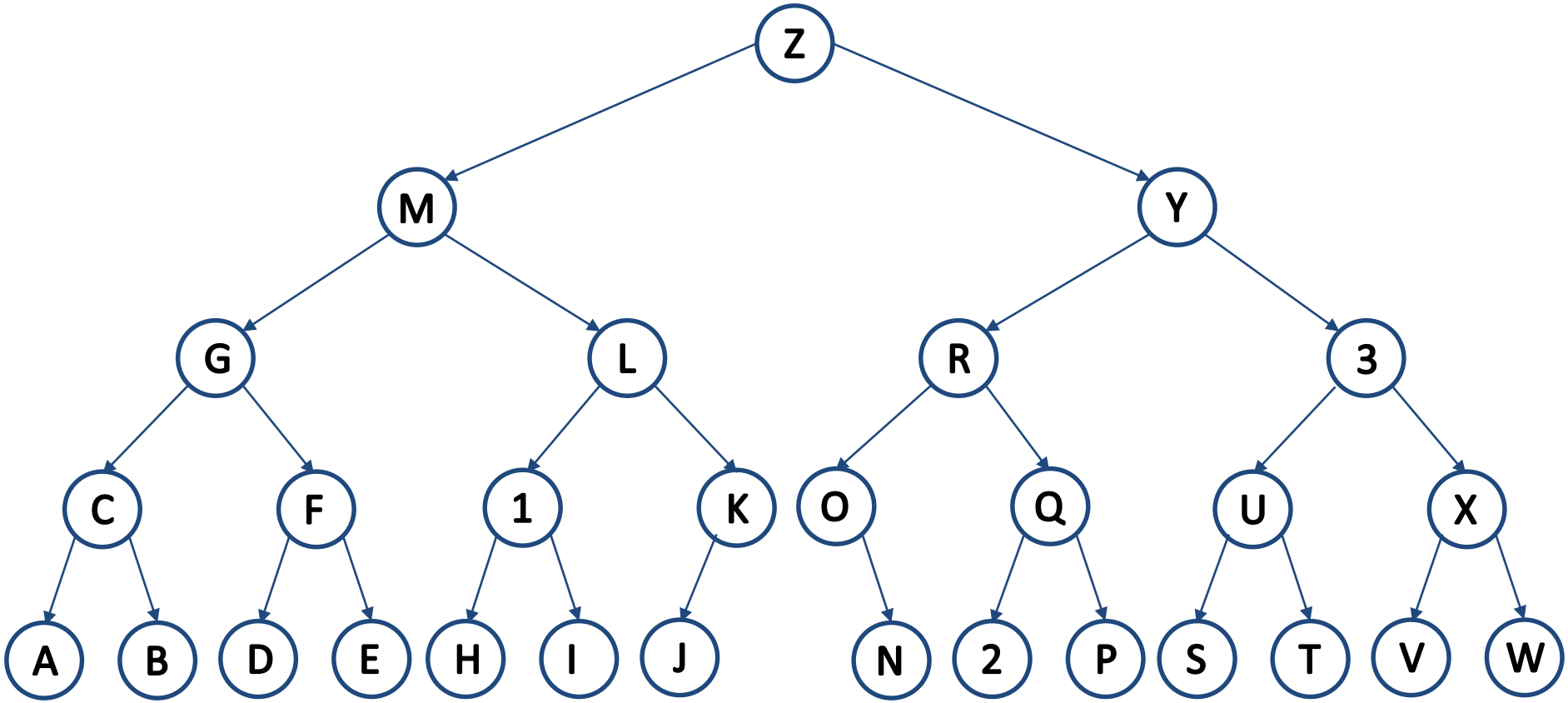


Z M G C A B F D E L 1 H I K J Y R O N Q 2 P 3 U S T X V W

Binary Tree In-Oder Traversal

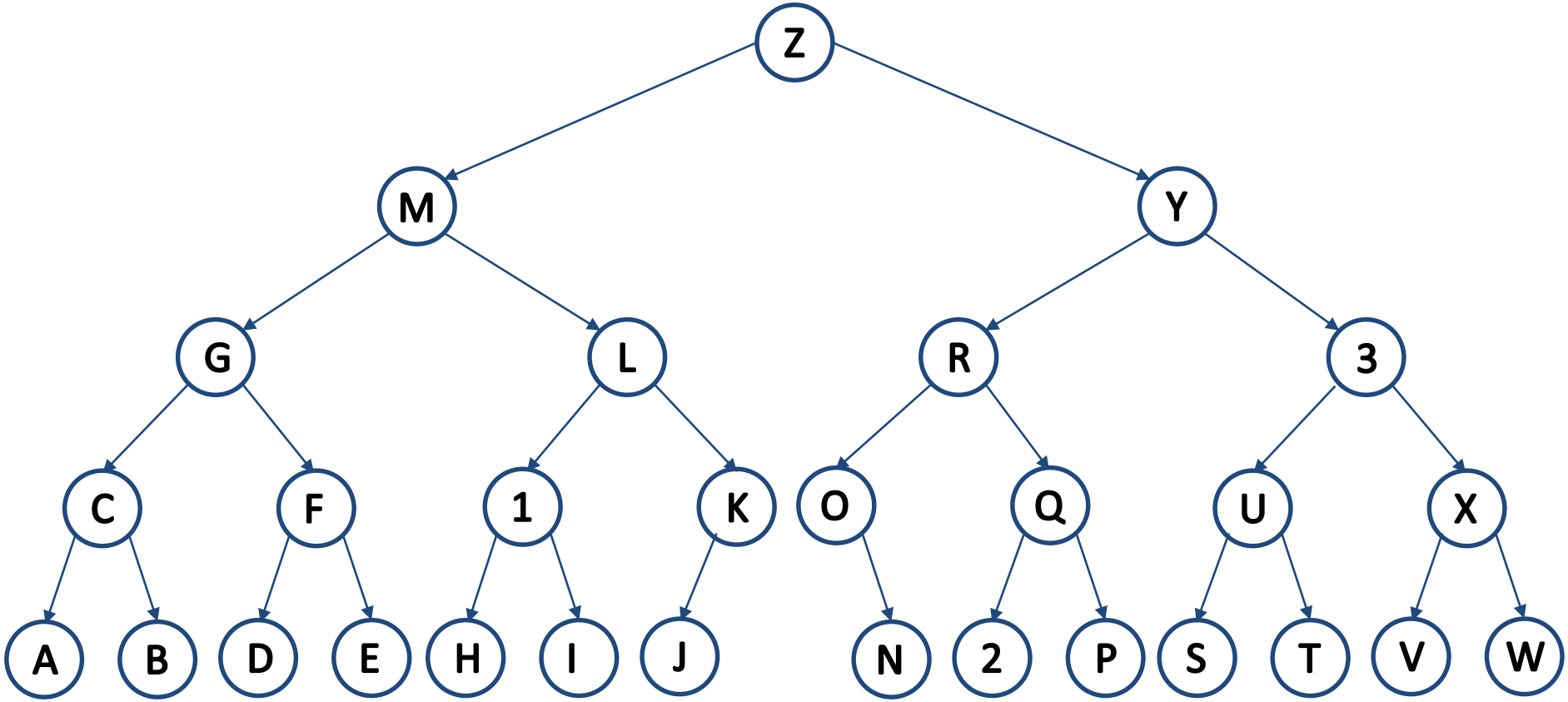


Binary Tree In-Oder Traversal

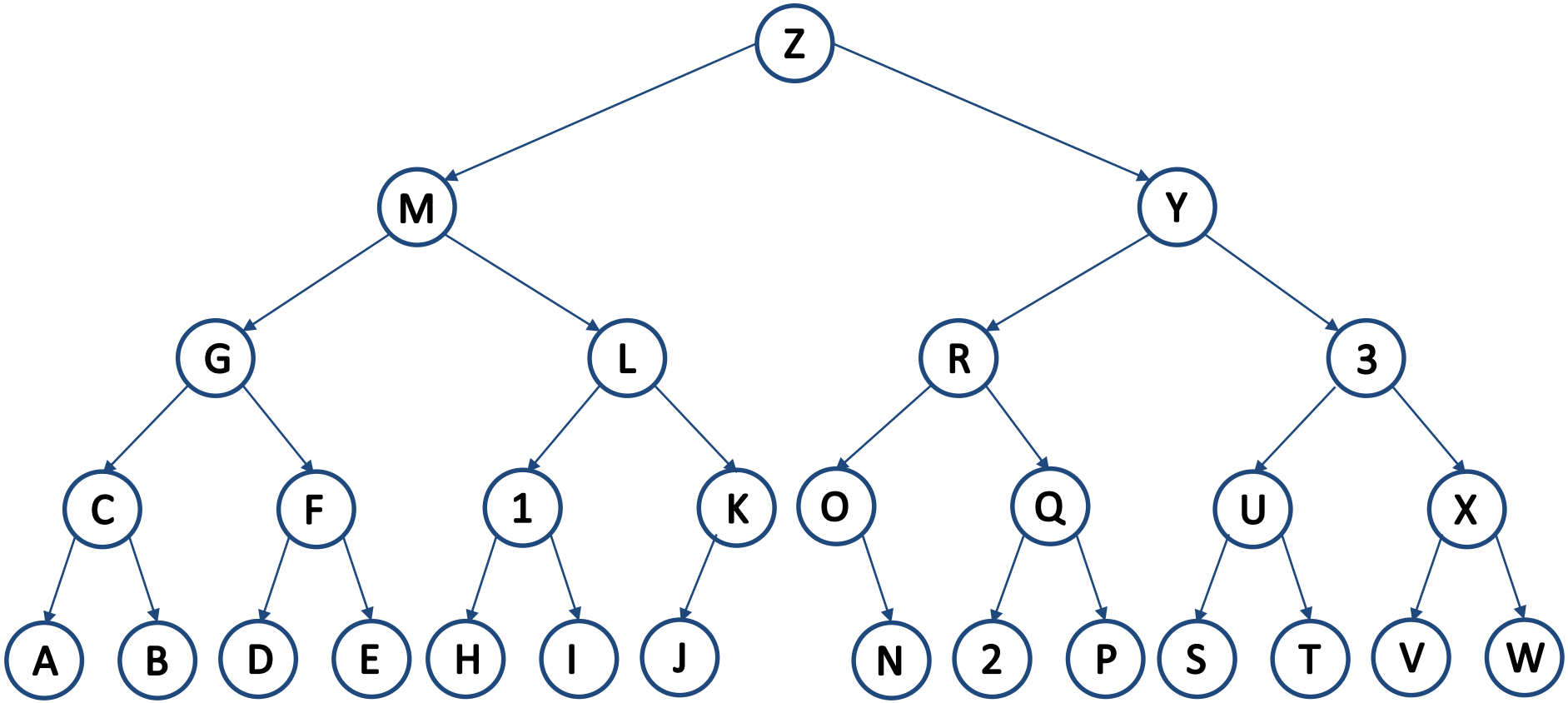


A C B G D F E M H 1 I L J K Z O N R 2 Q P Y S U T 3 V X W

Binary Tree Post-Oder Traversal

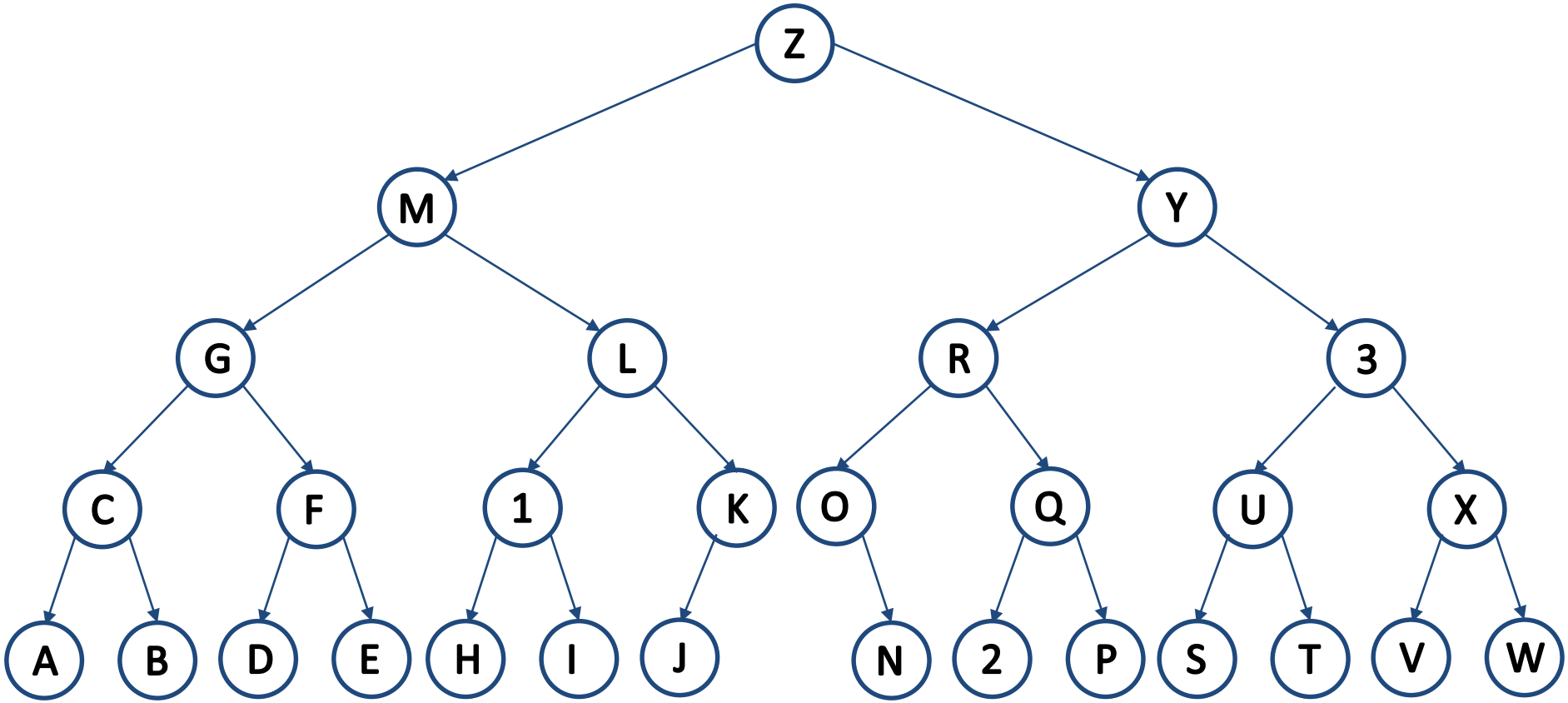


Binary Tree Post-Oder Traversal

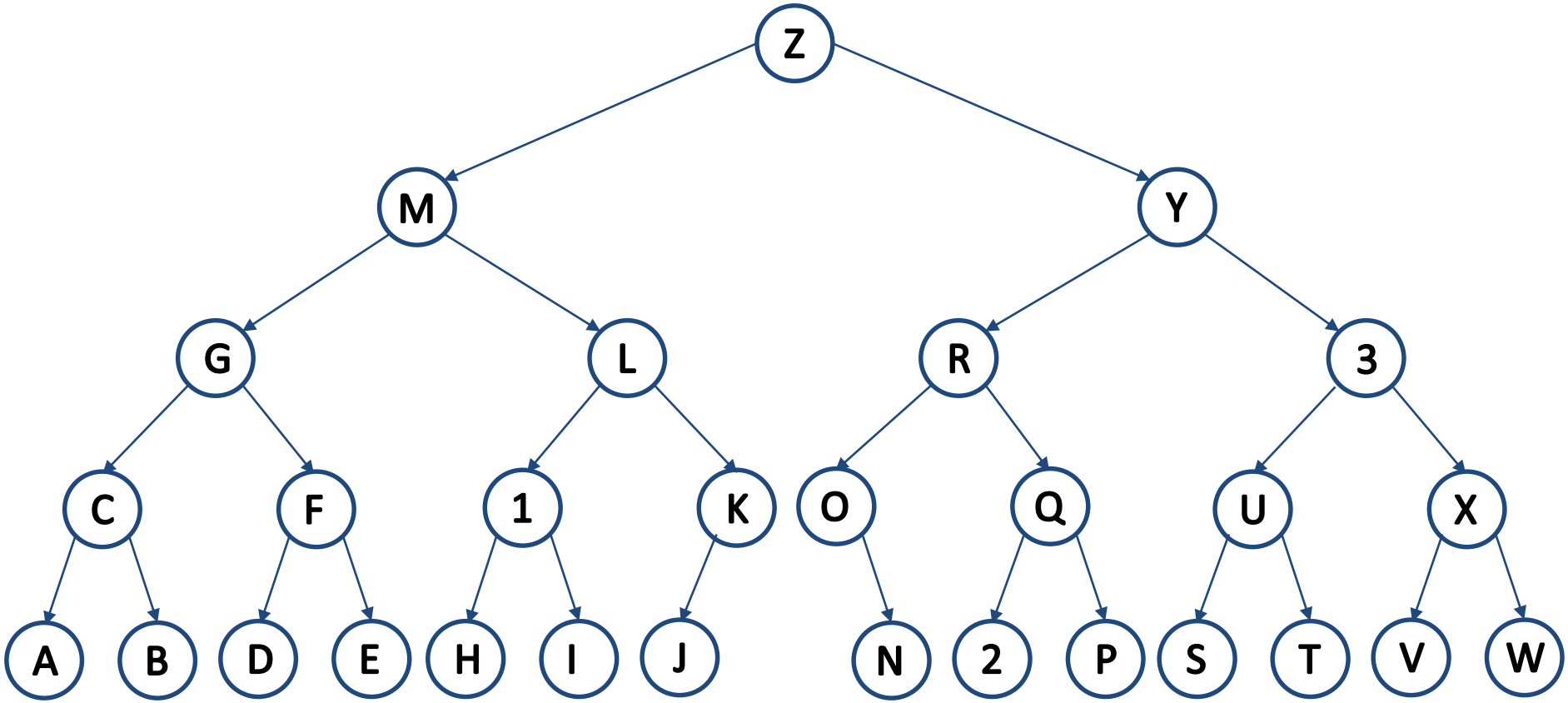


A B C D E F G H I 1 J K L M N O 2 P Q R S T U V W X 3 Y Z

Binary Tree Traversal (BFS)



Binary Tree Traversal (BFS)



Z M Y G L R 3 C F 1 K O Q U X A B D E H I J N 2 P S T V W