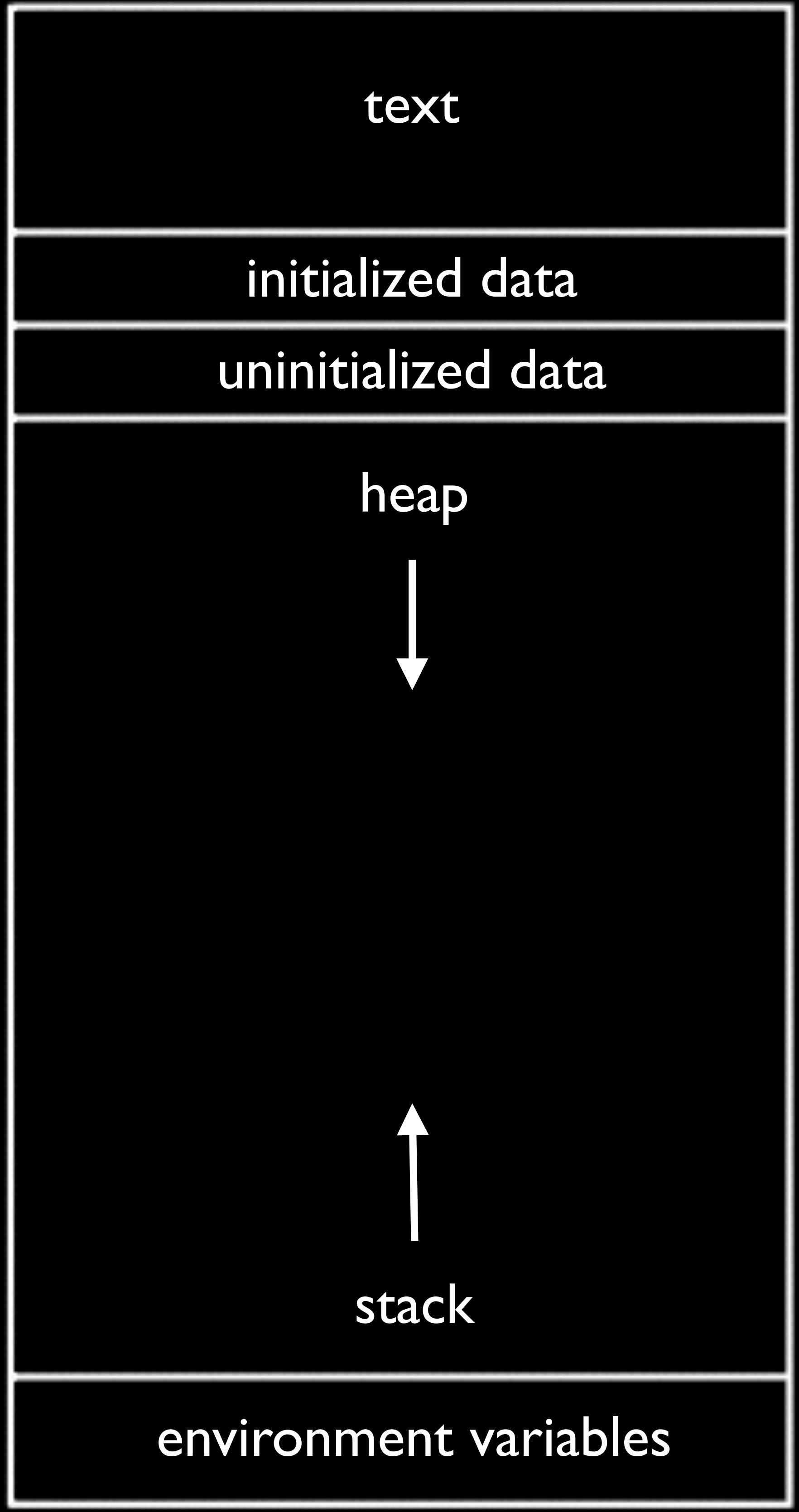
Week 5

last time

POINTER
FUN



text

initialized data

uninitialized data

heap

stack

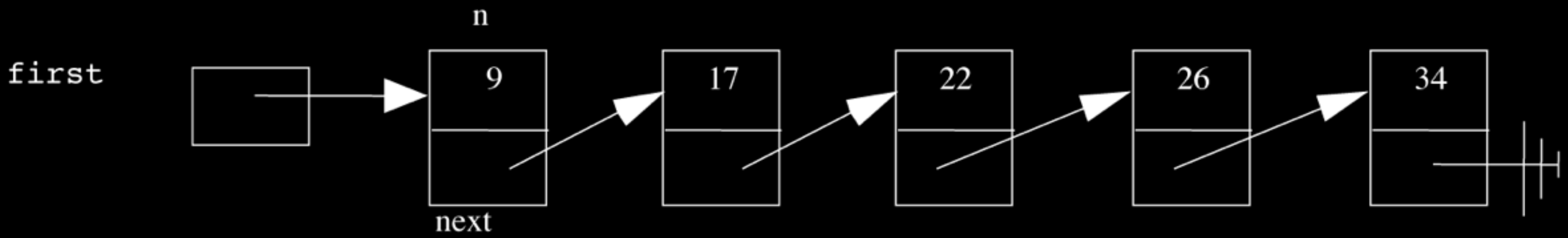
environment variables

```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

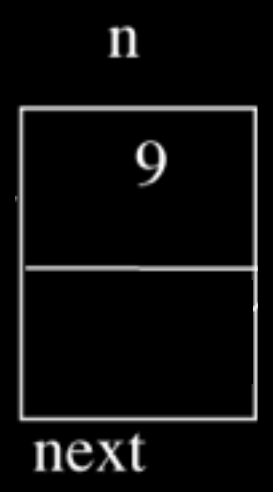
```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

```
typedef struct
{
    string name;
    string dorm;
}
student;
```

this time



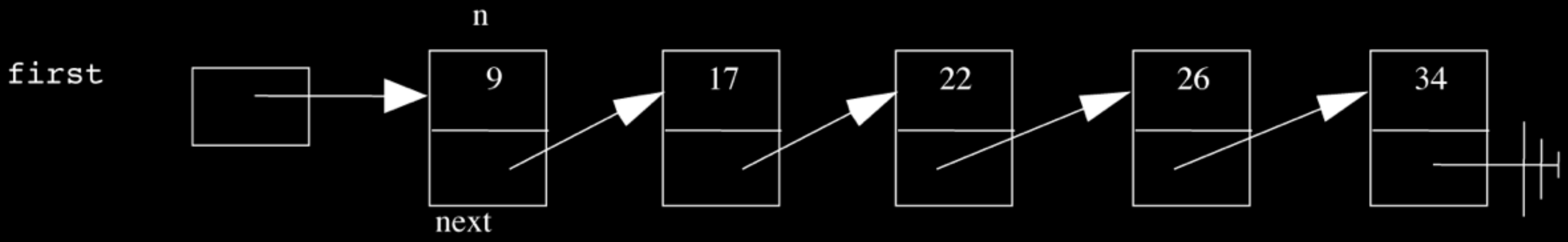
https://cs.calvin.edu/activities/books/c++/ds/1e/



http://cs.calvin.edu/books/c++/ds/1e/

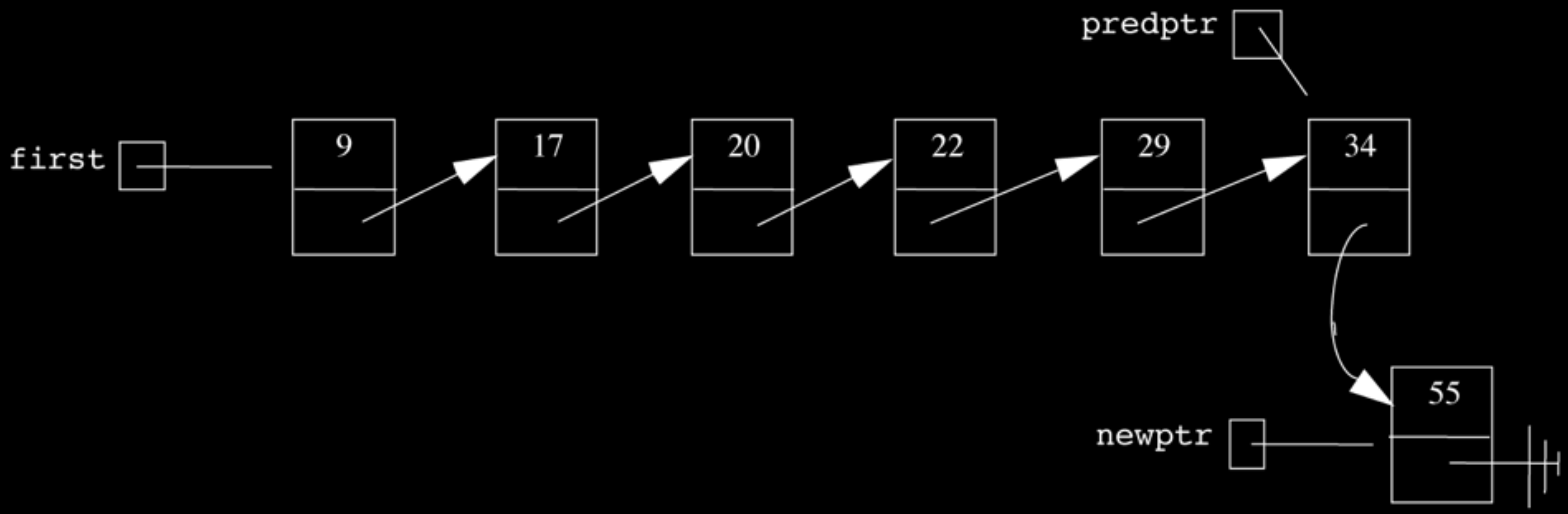
```
typedef struct node
{
    int n;
    struct node *next;
}
node;
```

```
delete

insert

search

...
```



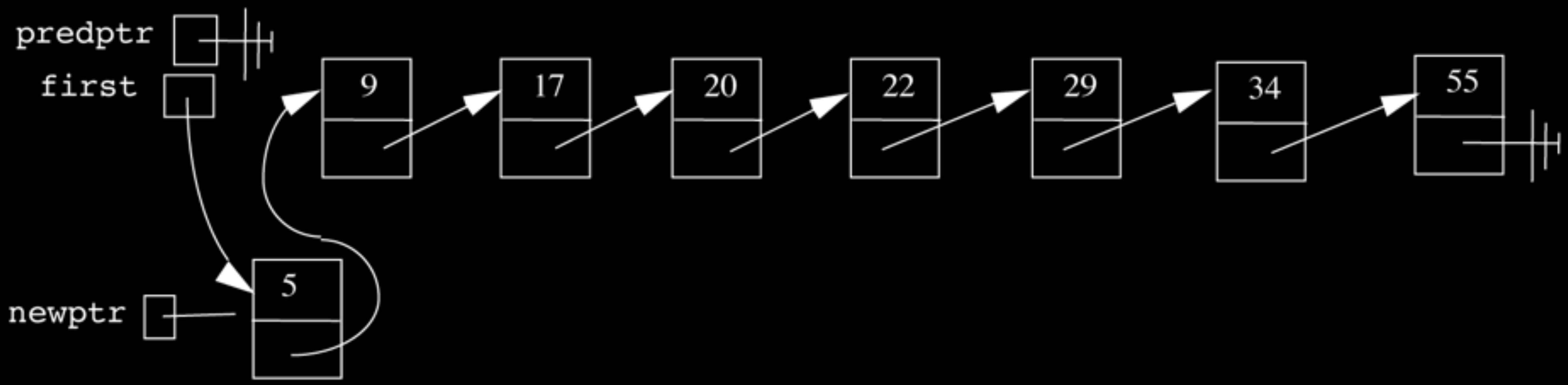
https://cs.calvin.edu/activities/books/c++/ds/1e/

insert at tail



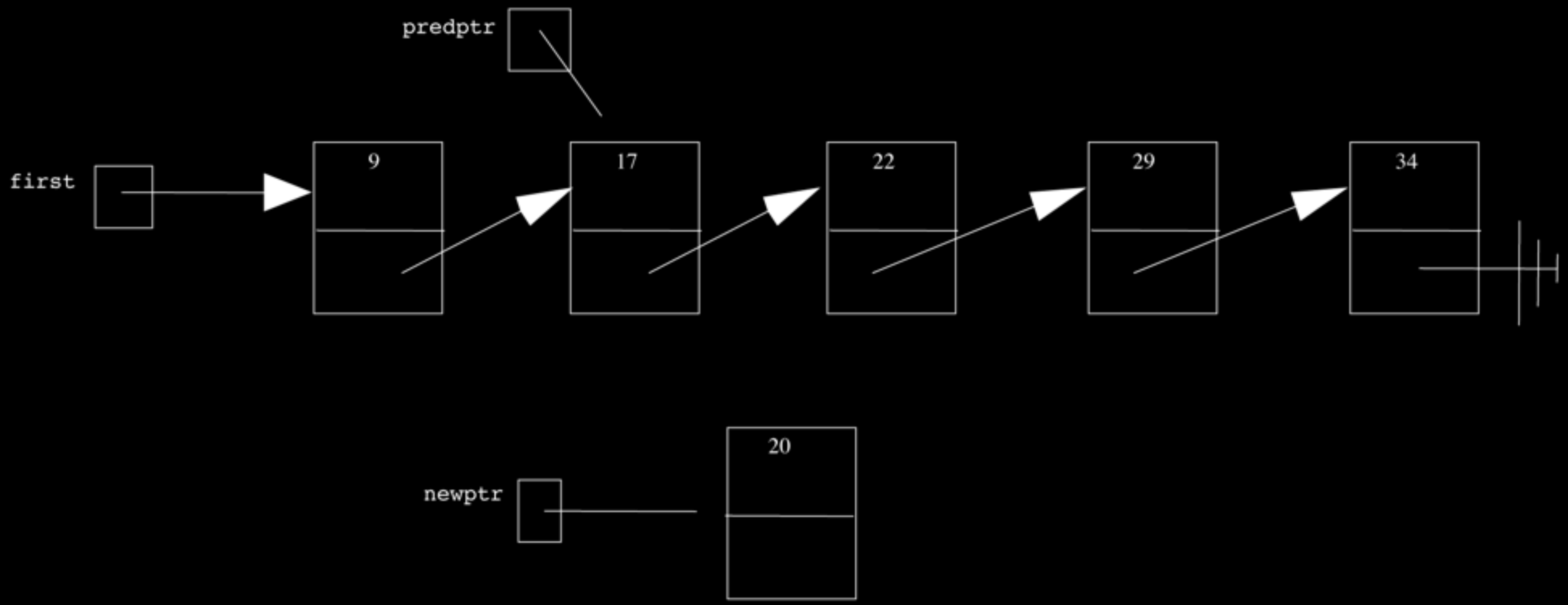
https://cs.calvin.edu/activities/books/c++/ds/1e/

insert at head



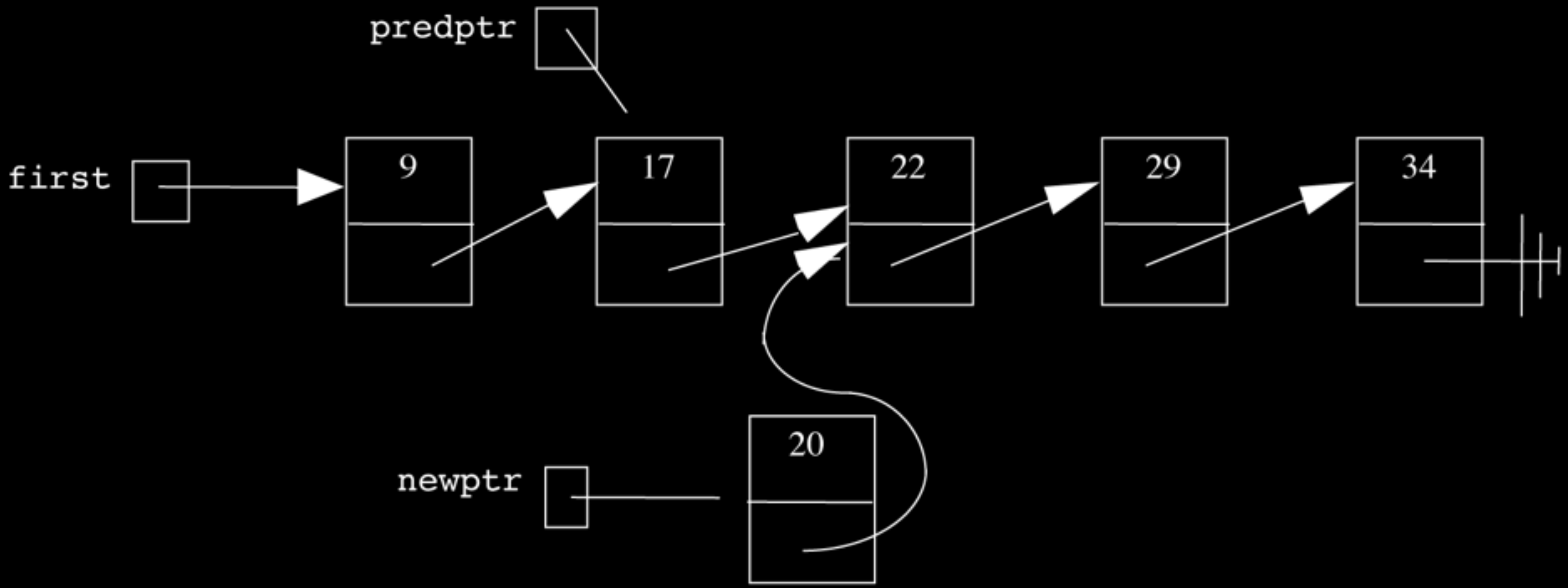
https://cs.calvin.edu/activities/books/c++/ds/1e/

insert in middle



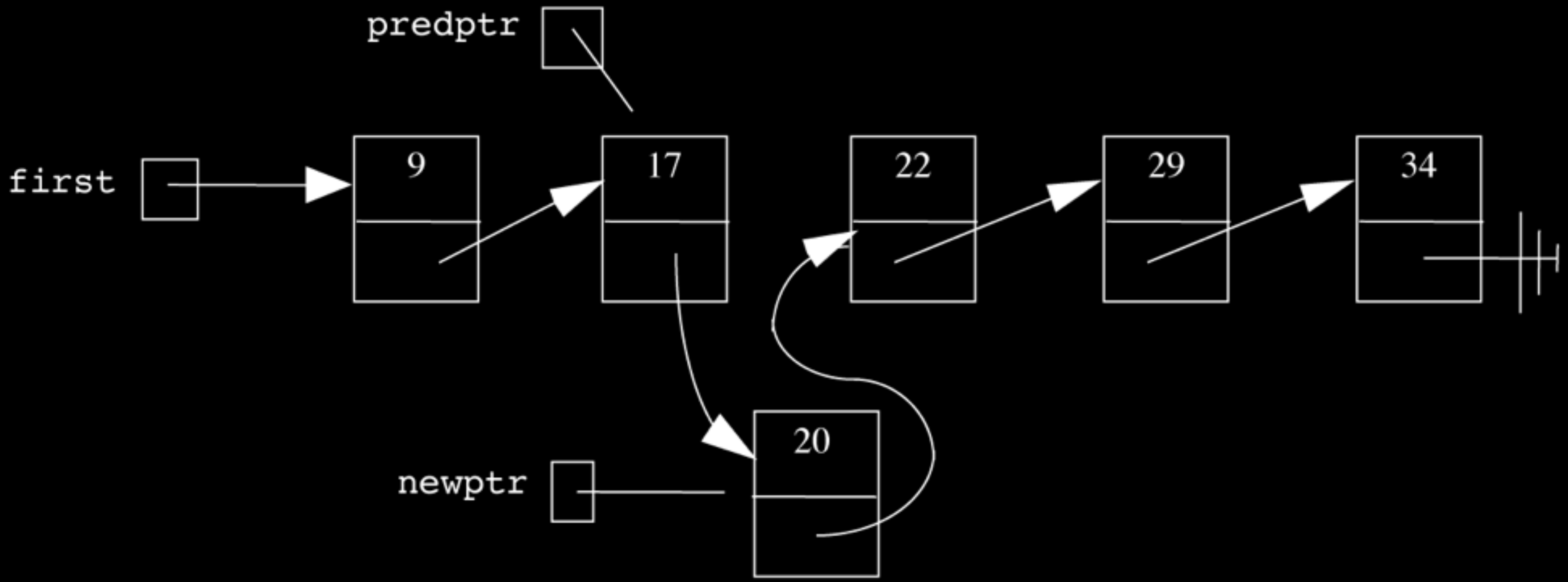
https://cs.calvin.edu/activities/books/c++/ds/1e/

insert in middle



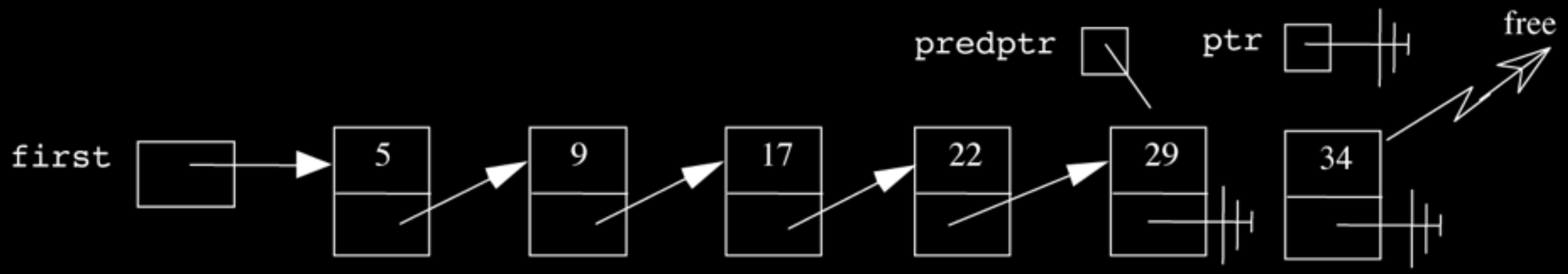
https://cs.calvin.edu/activities/books/c++/ds/1e/

insert in middle



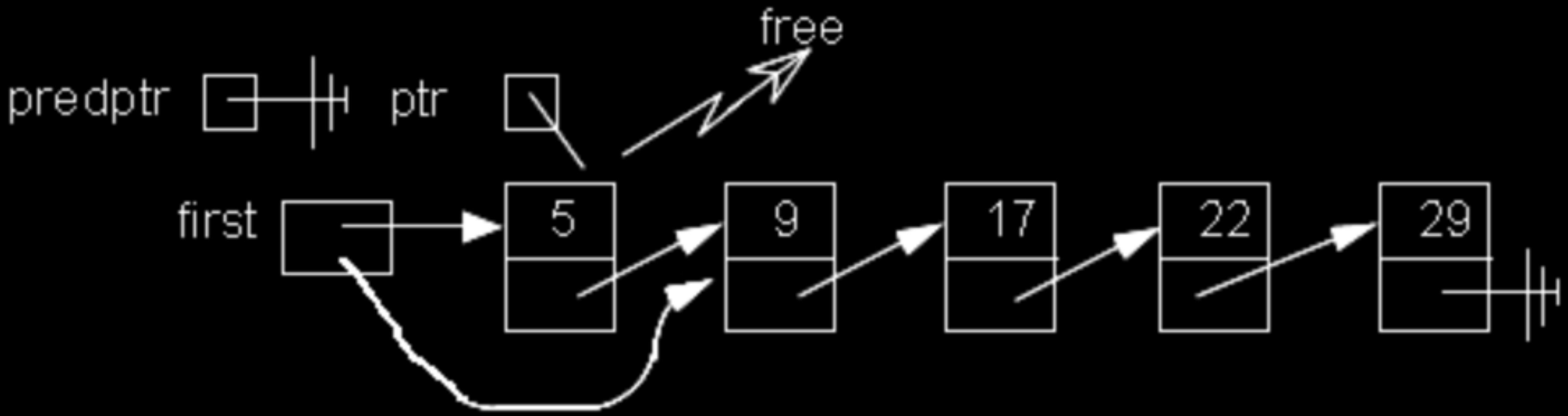
https://cs.calvin.edu/activities/books/c++/ds/1e/

remove tail



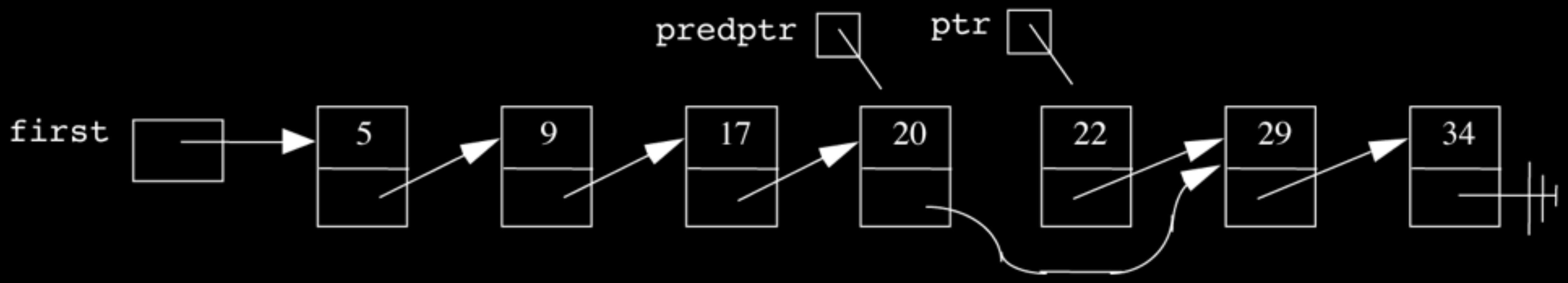
https://cs.calvin.edu/activities/books/c++/ds/1e/

remove head



https://cs.calvin.edu/activities/books/c++/ds/1e/

remove in middle



https://cs.calvin.edu/activities/books/c++/ds/1e/

delete

insert

search

...

```
bool search(int n, node *list)
{
    node *ptr = list;
    while (ptr != NULL)
    {
        if (ptr->n == n)
        {
            return true;
        }
        ptr = ptr->next;
    }
    return false;
}
```

SILVERWARE
PLATES ON A TRAY
DISHROOM TO PREVENT
EQUIPMENT PROBLEMS
THANKS FOR YOUR USE!
IS THE WAY

push

pop

...

```
typedef struct
{
    int numbers[CAPACITY];
    int size;
}
stack;
```

```
typedef struct
{
    int *numbers;
    int size;
}
stack;
```

engadget

enqueue

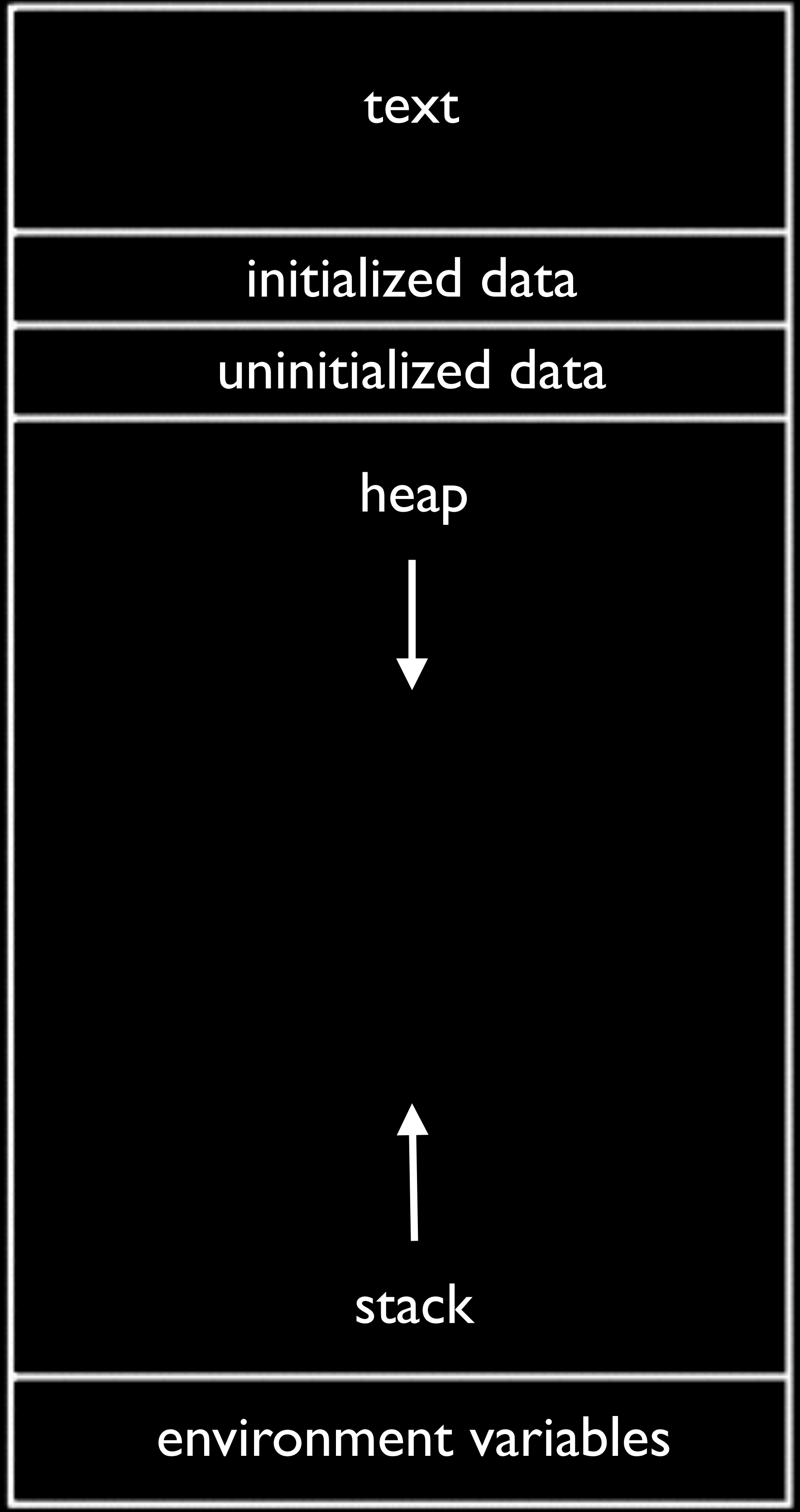
dequeue

. . .

```c
typedef struct
{
    int front;
    int numbers[CAPACITY];
    int size;
}
queue;
```

```
typedef struct
{
    int front;
    int *numbers;
    int size;
}
queue;
```

Jack Learns the Facts About Queues and Stacks



text

initialized data

uninitialized data

heap

stack

environment variables

tree

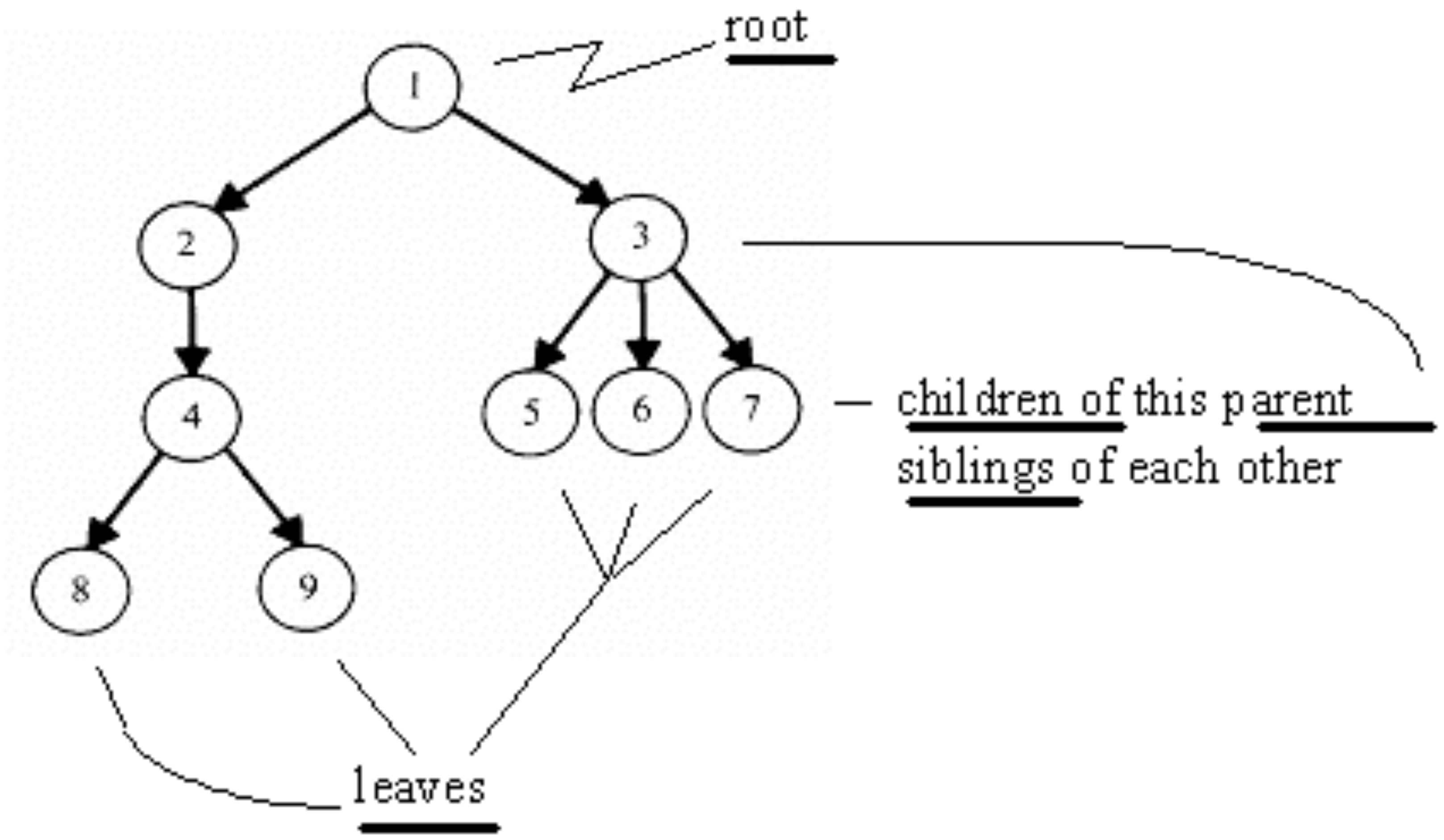


Figure by Larry Nyhoff.

22 33 44 55 66 77 88

22	33	44	55	66	77	88

binary search tree

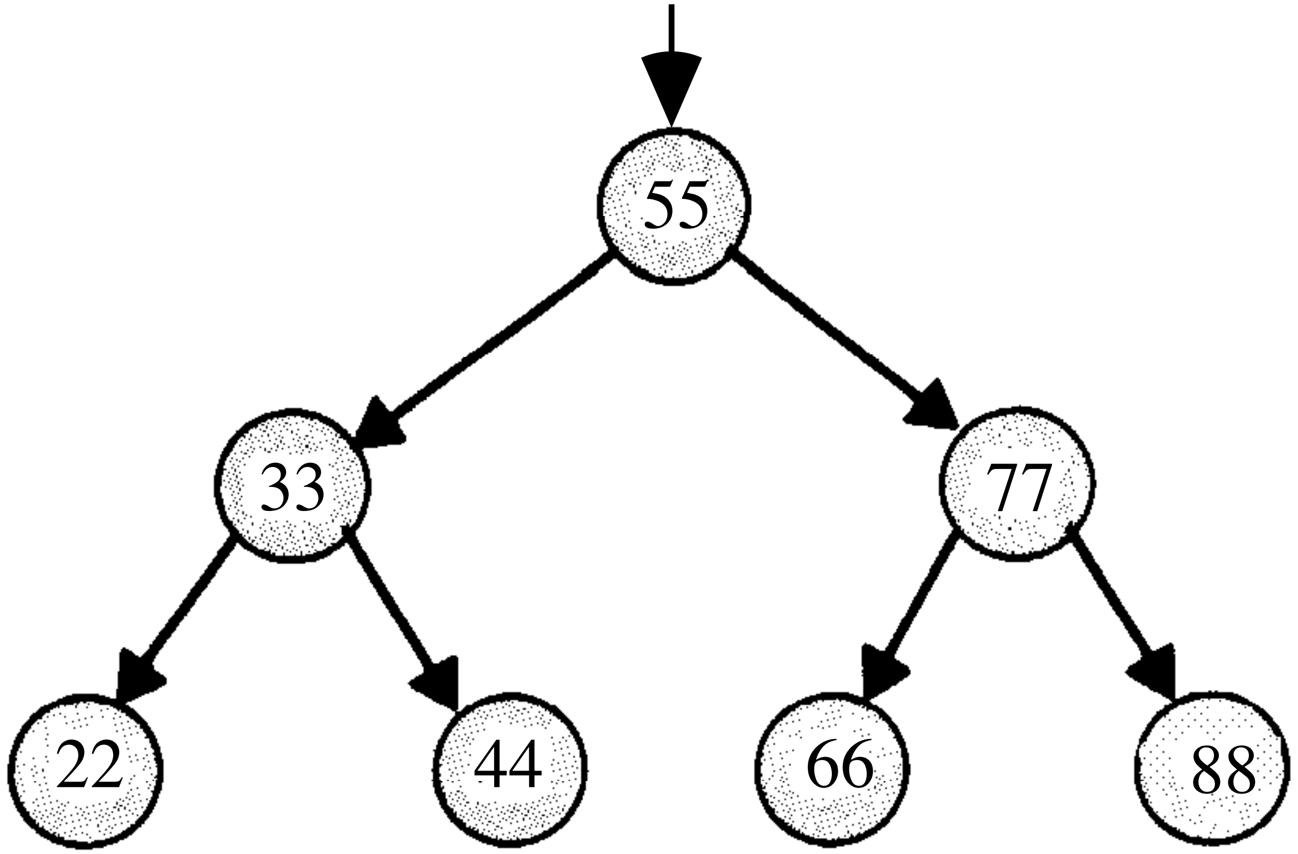


Figure from http://cs.calvin.edu/books/c++/ds/1e/.

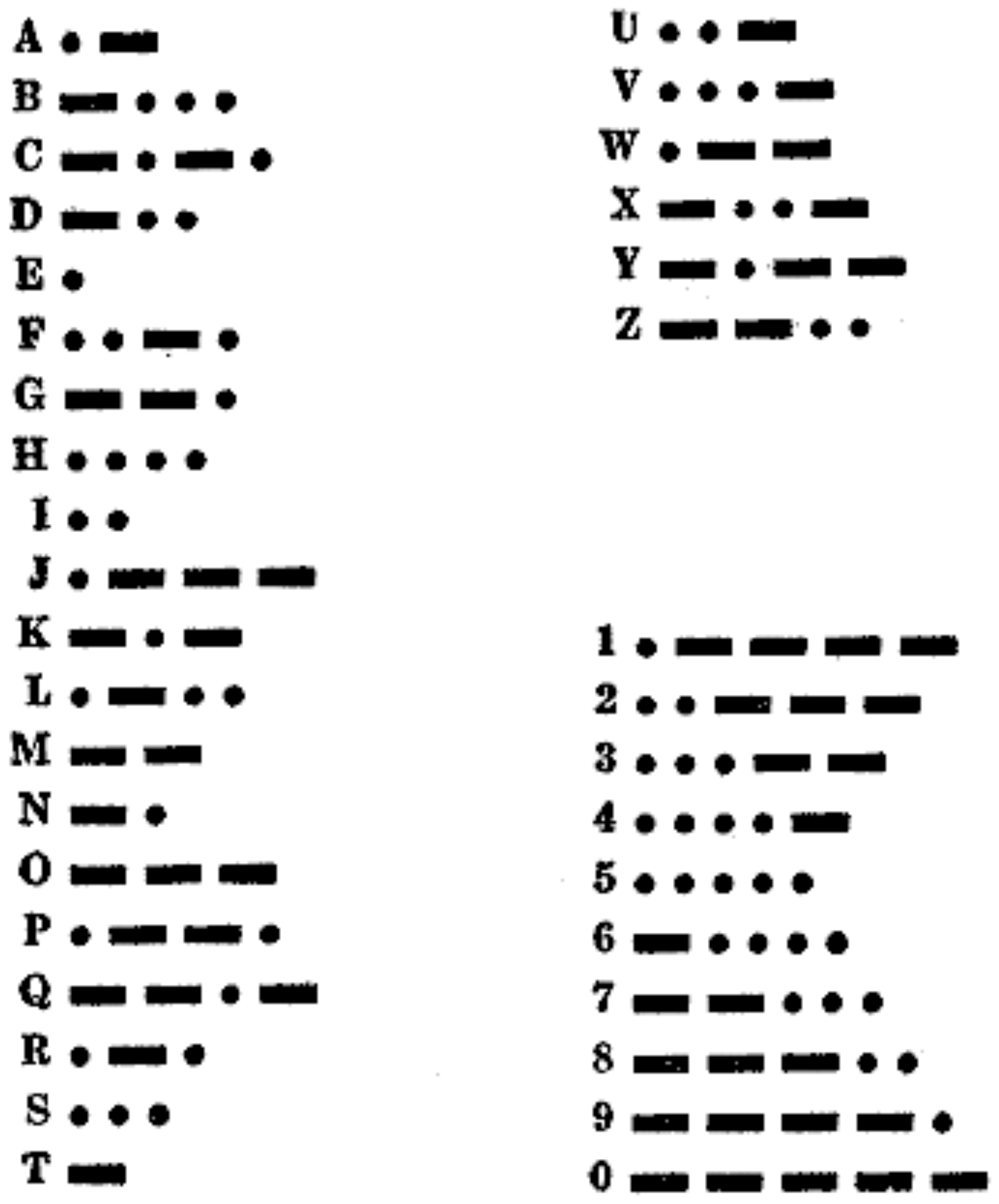
```
typedef struct node
{
    int n;
    struct node *left;
    struct node *right;
}
node;
```

```
bool search(int n, node *tree)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (n < tree->n)
    {
        return search(n, tree->left);
    }
    else if (n > tree->n)
    {
        return search(n, tree->right);
    }
    else
    {
        return true;
    }
}
```

ASCII

A	B	C	D	E	F	G	H	I	...
65	66	67	68	69	70	71	72	73	...

morse code



A	● ▬
B	▬ ● ● ●
C	▬ ● ▬ ●
D	▬ ● ●
E	●
F	● ● ▬ ●
G	▬ ▬ ●
H	● ● ● ●
I	● ●
J	● ▬ ▬ ▬
K	▬ ● ▬
L	● ▬ ● ●
M	▬ ▬
N	▬ ●
O	▬ ▬ ▬
P	● ▬ ▬ ●
Q	▬ ▬ ● ▬
R	● ▬ ●
S	● ● ●
T	▬
U	● ● ▬
V	● ● ● ▬
W	● ▬ ▬
X	▬ ● ● ▬
Y	▬ ● ▬ ▬
Z	▬ ▬ ● ●
1	● ▬ ▬ ▬ ▬
2	● ● ▬ ▬ ▬
3	● ● ● ▬ ▬
4	● ● ● ● ▬
5	● ● ● ● ●
6	▬ ● ● ● ●
7	▬ ▬ ● ● ●
8	▬ ▬ ▬ ● ●
9	▬ ▬ ▬ ▬ ●
0	▬ ▬ ▬ ▬ ▬

Image adapted from Wikipedia.

“ECEABEADCAEDEEEECEADEEEEEEDBAAEABDBBAAEAAAC
DDCCEABEEDCBEEDEAEEEEEAEEDBCEBEEADEAEEDAEBC
DEDEAEEDCEEAEEE”

“ECEABEADCAEDEEEECEADEEEEEEDBAAEABDBBAAEAAAC
DDCCEABEEDCBEEDEAEEEEEAEEDBCEBEEADEAEEDAEBC
DEDEAEEDCEEAEEE”

character	**A**	**B**	**C**	**D**	**E**
frequency	0.2	0.1	0.1	0.15	0.45

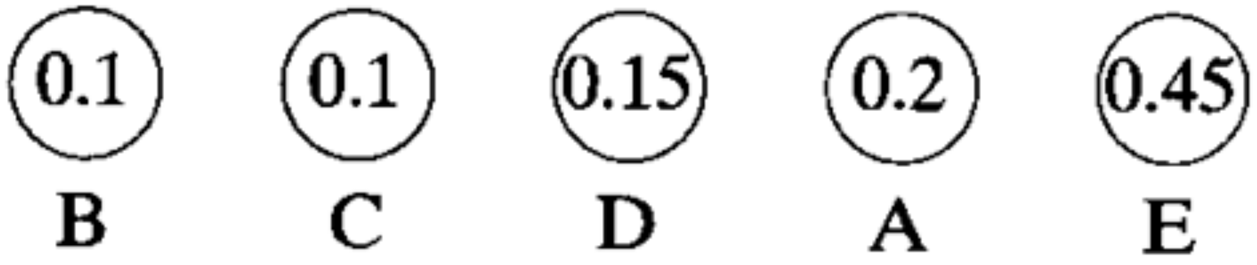


Figure by Larry Nyhoff.

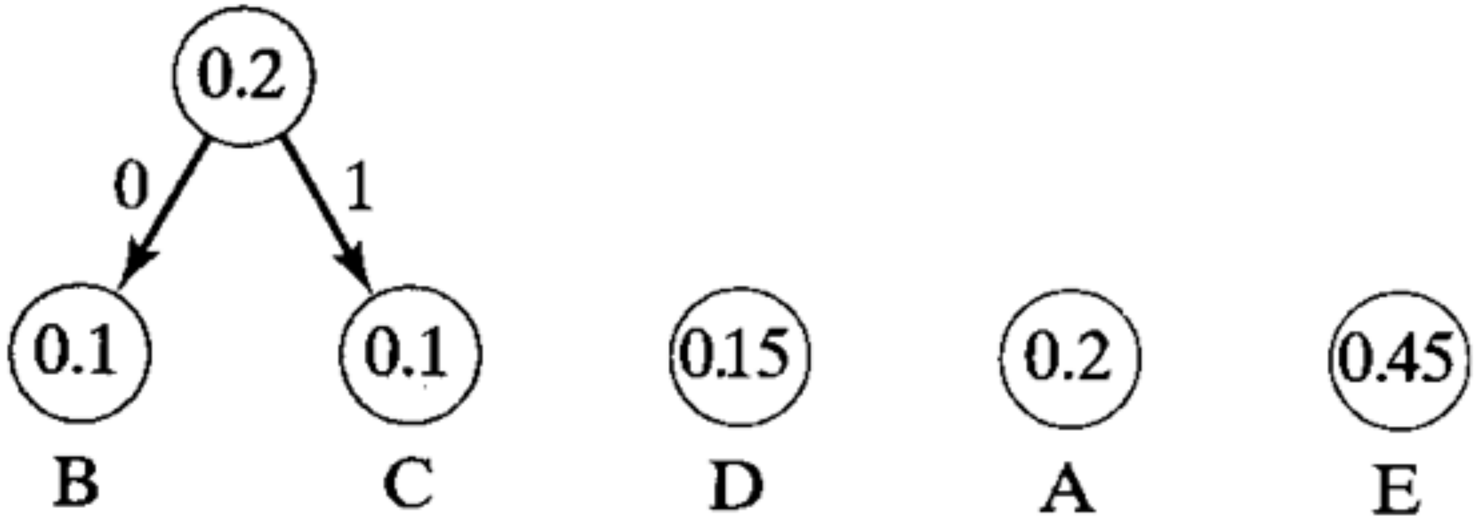


Figure by Larry Nyhoff.

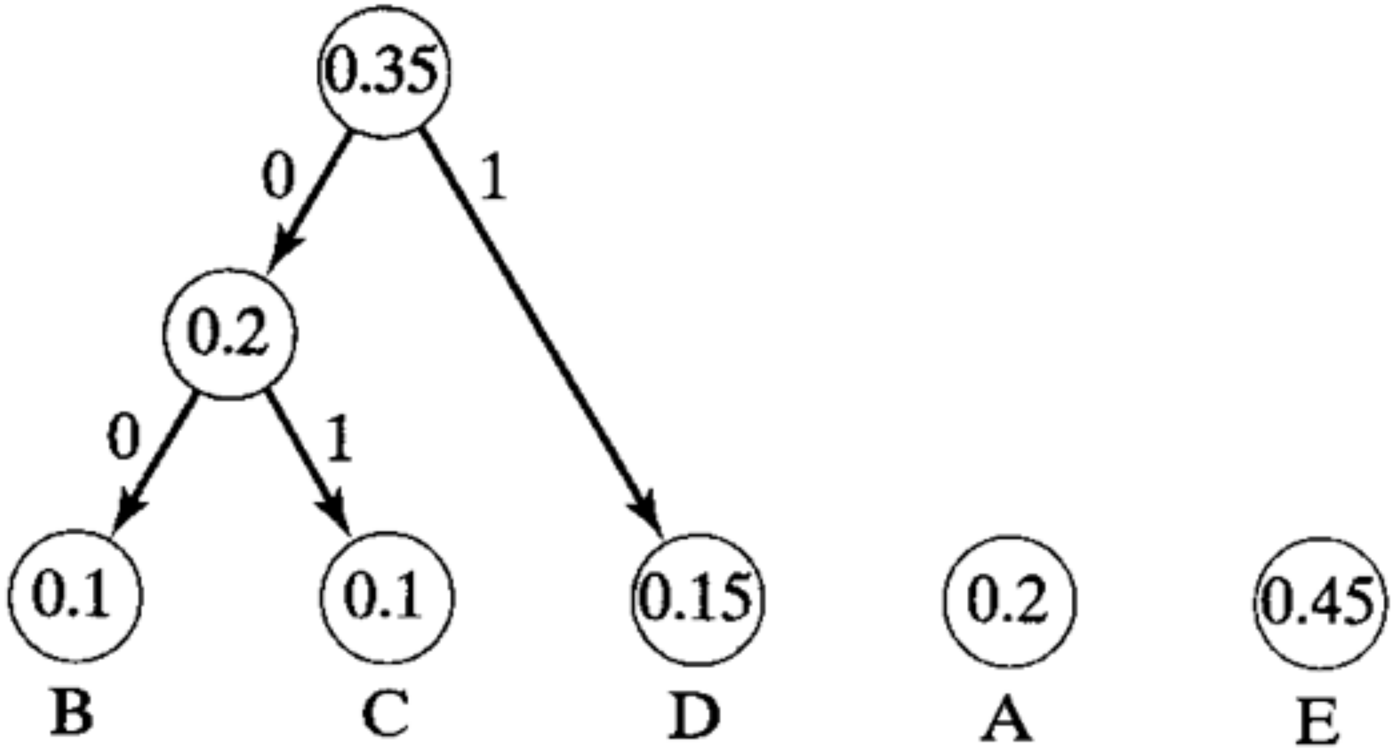


Figure by Larry Nyhoff.

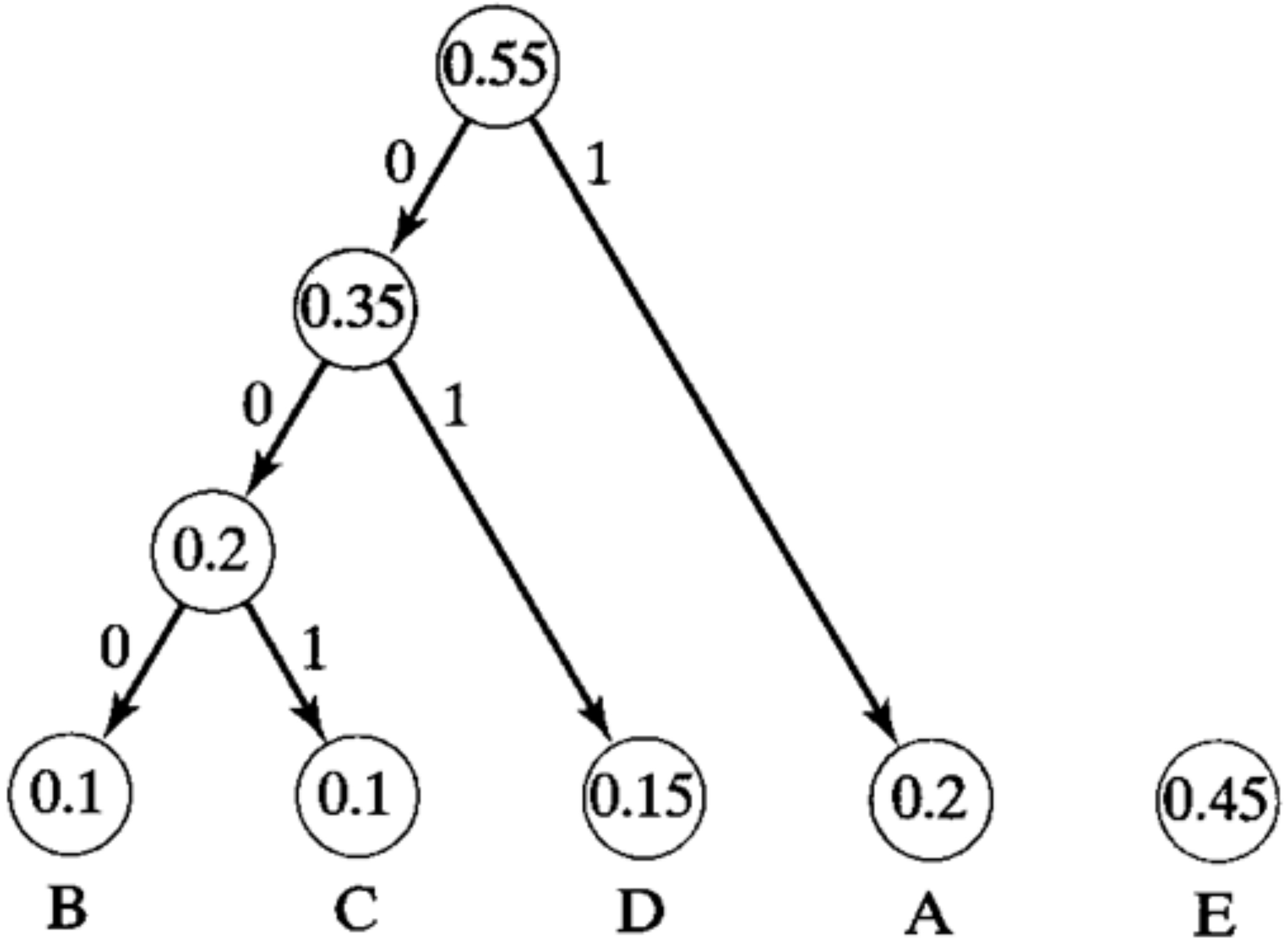


Figure by Larry Nyhoff.

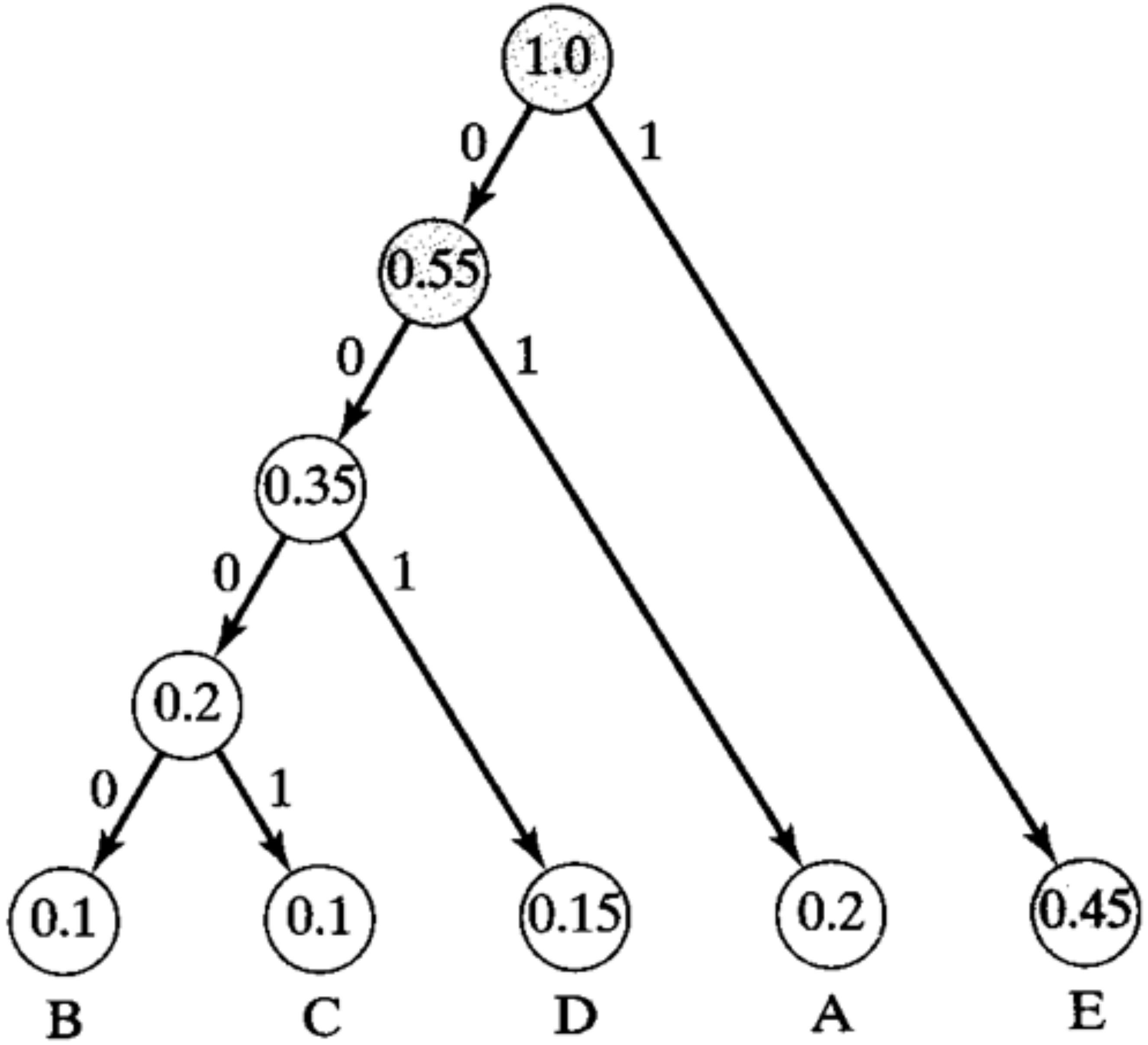


Figure by Larry Nyhoff.

A is 01

B is 0000

C is 0001

D is 001

E is 1

```
typedef struct node
{
    char symbol;
    float frequency;
    struct node *left;
    struct node *right;
}
node;
```

…

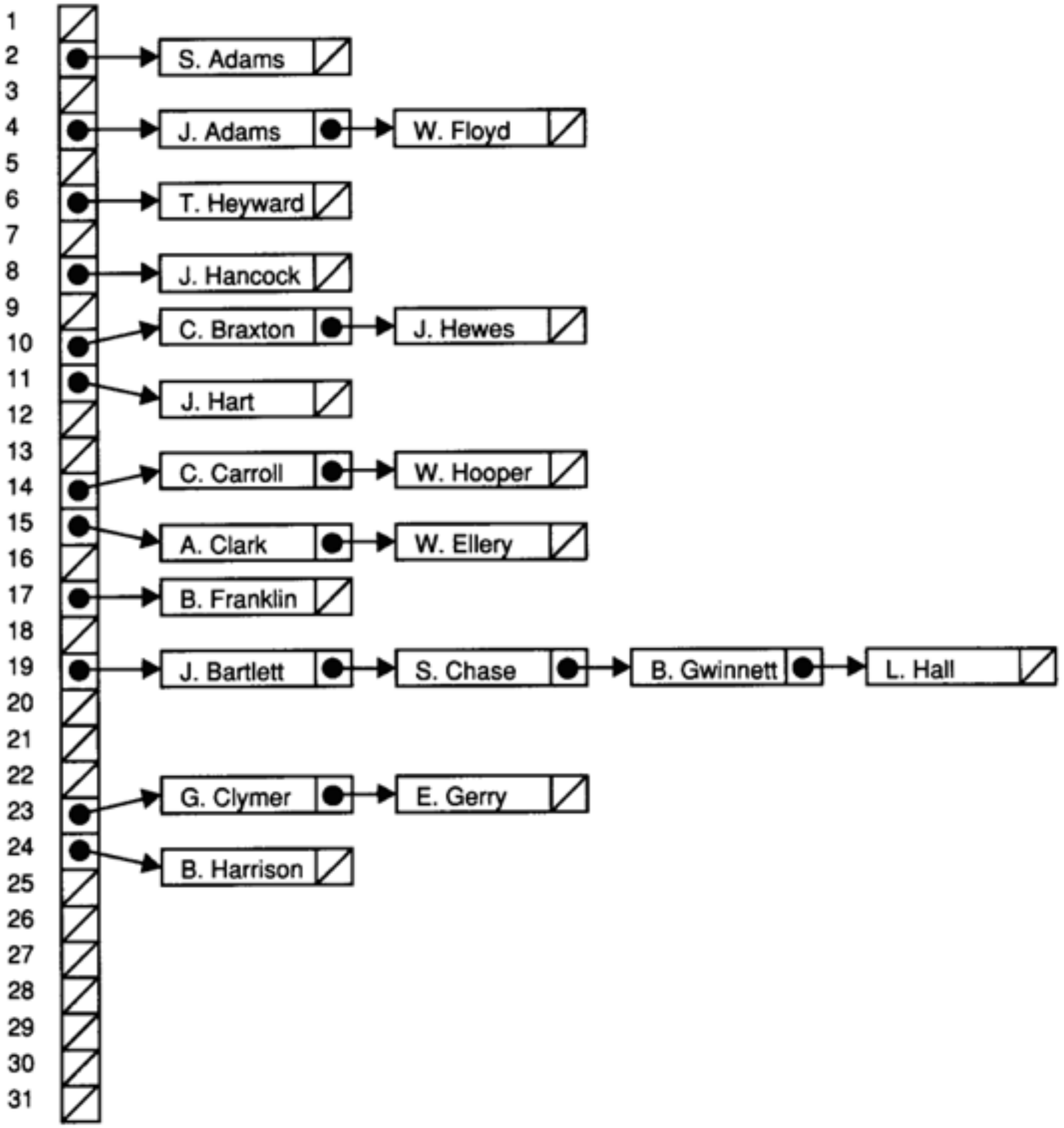
$O(n)$

$O(\log n)$

$O(1)$

…

table[0]	
table[1]	
table[2]	
table[3]	
table[4]	
table[5]	
table[6]	
	⋮
table[n-1]	


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
S. Adams
J. Adams
W. Floyd
T. Heyward
J. Hancock
C. Braxton
J. Hewes
J. Hart
C. Carroll
W. Hooper
A. Clark
W. Ellery
B. Franklin
J. Bartlett
S. Chase
B. Gwinnett
L. Hall
G. Clymer
E. Gerry
B. Harrison

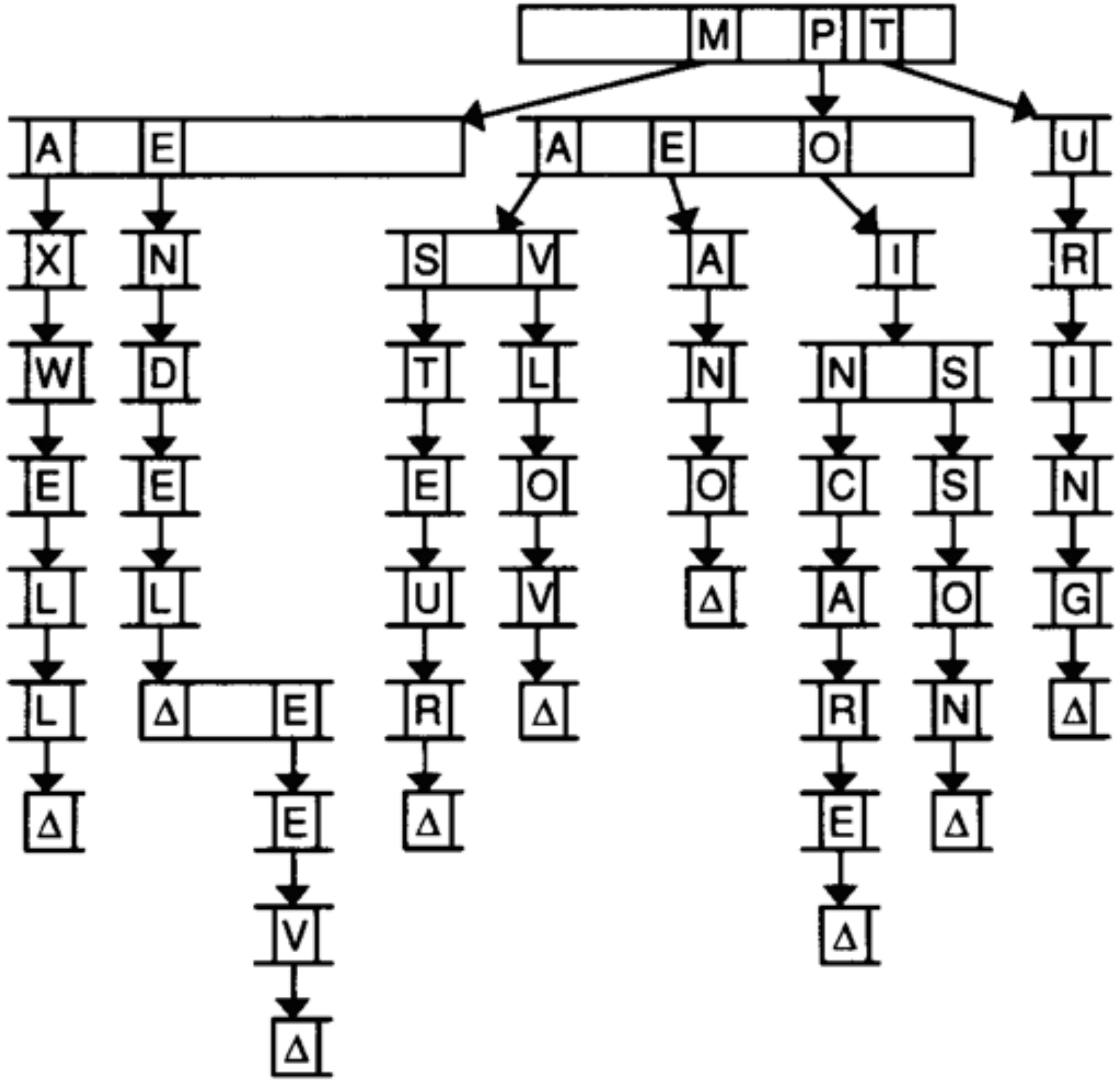


Figure from Lewis and Denenberg's Data Structures & Their Algorithms.

```
typedef struct node
{
    bool word;
    struct node *children[27];
}
node;
```

Week 5