CSE 251 - struct and union

Arend Hintze

	L2	L3	L4	L5	L6	L8
AH	8	10	10	0	0	0
AN	8	8	10	0	6	0
AR	8	8	8	9	7	10
CR	10	10	10	4	8	10
DK	10	5	10	4	0	10
DM	10	8	10	10	7	10
DS	10	10	10	10	0	10
DZ	10	8	10	10	8	10
FL	8	5	10	10	7	9
GJ	8	8	10	10	0	8
GW	10	10	10	10	10	8
HK	8	10	0	10	9	8
JG	10	9	7	10	7	7
JL	9	9	10	10	10	7
JM	10	5	8	10	8	8
JS	10	10	10	6	8	8
JW	10	10	8	10	9	8
KW	10	8	8	10	9	10
MJ	10	8	7	10	10	10
MR	10	10	10	10	9	0
MS	8	10	8	8	8	10
NH	8	10	0	0	8	8
RB	0	10	10	10	10	8
RL	10	8	8	10	10	8
RS	10	10	8	4	9	0
SG	10	8	10	10	0	0
TP	10	8	7	10	6	7
TS	10	10	8	10	8	0
XZ	8	5	10	10	6	8
YJ	10	10	8	10	10	6

homework

- load a file specified by argv[1]
- find out how often each word appears
- show that list

assumptions

- words are separated by <space>
- there can not be more unique words than words in total
- the data structure must “grow”
- the DS must count the words
- the DS must tell if word exists or not

Count words:

```
F=fopen(argv[1],"r");
while(!feof(F)){
    fscanf(F,"%s",word);
    wordCount++;
}
rewind(F);
```

Create Data structure:

```
List=(char**)malloc(sizeof(char*)*wordCount);
count=(int*)malloc(sizeof(int*)*wordCount);
for(i=0;i<wordCount;i++)
    count[i]=0;
i=0;
```

Sort words in:

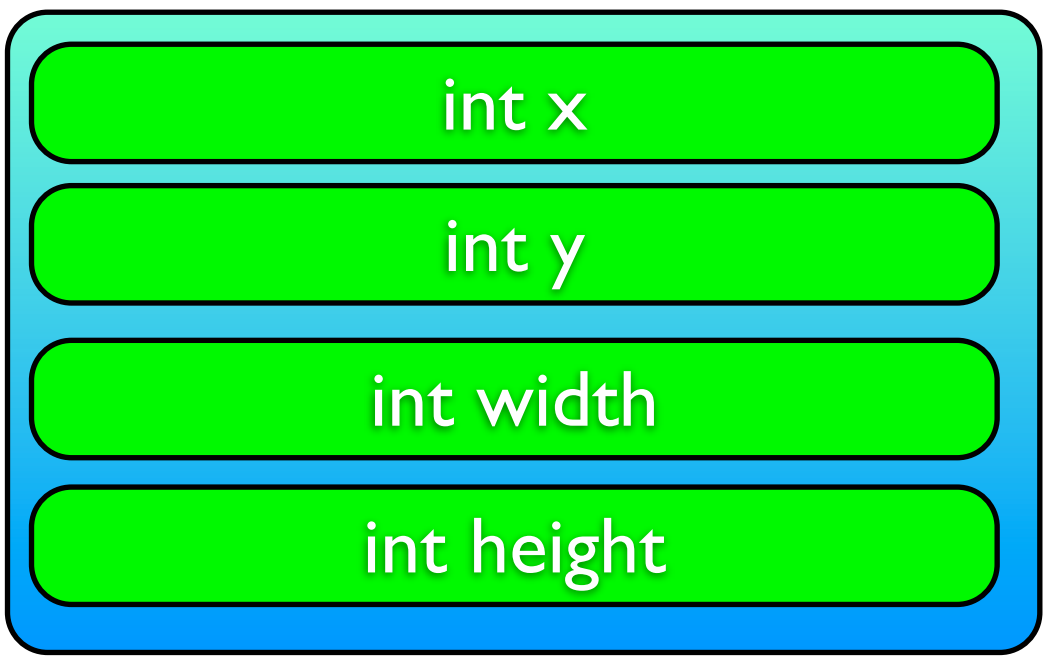
```
i=0;
while(!feof(F)){
    fscanf(F,"%s",word);
    b=true;
    for(j=0;j<i;j++)
        if(strcmp(List[j],word)==0){
            //printf("%s word exists\n",word);
            count[j]++;
            b=false;
        }
    if(b){
        //is word is not in list
        List[i]=(char*)malloc(sizeof(char)*(strlen(word)+1));
        count[i]=1;
        strcpy(List[i],word);
        i++;
    }
}
```

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#include <string.h>

int main(int argc,const char* argv[]){
    FILE *F;
    char word[100];
    int i,j,wordCount=0;
    char **List;
    int *count;
    bool b;
    F=fopen(argv[1],"r");
    while(!feof(F)){
        fscanf(F,"%s",word);
        wordCount++;
    }
    rewind(F);

    List=(char**)malloc(sizeof(char*)*wordCount);
    count=(int*)malloc(sizeof(int*)*wordCount);
    for(i=0;i<wordCount;i++)
        count[i]=0;

    i=0;
    while(!feof(F)){
        fscanf(F,"%s",word);
        b=true;
        for(j=0;j<i;j++)
            if(strcmp(List[j],word)==0){
                //printf("%s word exists\n",word);
                count[j]++;
                b=false;
            }
        if(b){
            //is word is not in list
            List[i]=(char*)
              malloc(sizeof(char)*(strlen(word)+1));
            count[i]=1;
            strcpy(List[i],word);
            i++;
        }
    }
    for(j=0;j<i;j++)
        printf("%i %i %s\n",j,count[j],List[j]);
    printf("wordcount: %i\n",wordCount);
    fclose(F);
      return 0;
}
```

There are better ways to do that!

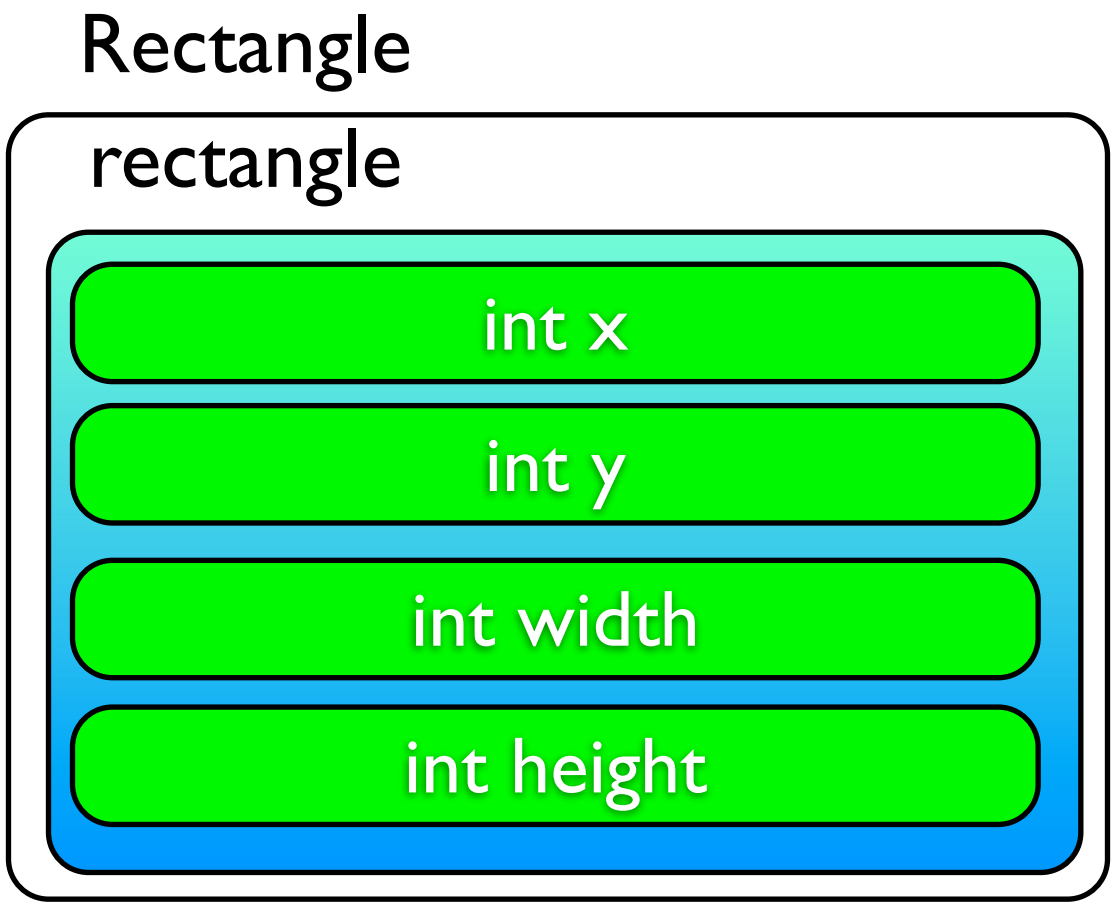
```c
#include <stdio.h>
#include <stdlib.h>

struct rectangle{
    int x,y,width,height;
};

int main(int argc,const char* argv[]){
    struct rectangle myRect;
    myRect.x=0;
    myRect.y=0;
    myRect.width=10;
    myRect.height=5;
}
```

rectangle



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

struct rectangle{
    int x,y,width,height;
};

void showRect(struct rectangle theRect);

int main(int argc,const char* argv[]){
    struct rectangle myRect;
    myRect.x=0;
    myRect.y=0;
    myRect.width=10;
    myRect.height=5;
    showRect(myRect);
}

void showRect(struct rectangle theRect){
    printf("X:%i Y:%i width:%i height:%i
\n",theRect.x,theRect.y,theRect.width,theRect.height);
}
```

We can turn a struct into a type



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

typedef struct rectangle{
    int x,y,width,height;
} Rectangle;

void showRect(Rectangle theRect);

int main(int argc,const char* argv[]){
    Rectangle myRect;
    myRect.x=0;
    myRect.y=0;
    myRect.width=10;
    myRect.height=5;
    showRect(myRect);
    printf("%i\n",(int)sizeof(Rectangle));
}

void showRect(Rectangle theRect){
    printf("X:%i Y:%i width:%i height:%i\n",theRect.x,theRect.y,theRect.width,theRect.height)
}
```

(*myRect).x is synonymous with myRect->x

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

typedef struct rectangle{
    int x,y,width,height;
} Rectangle;

void showRect(Rectangle theRect);

int main(int argc,const char* argv[]){
    Rectangle *myRect;
    myRect=(Rectangle*)malloc(sizeof(Rectangle));
    myRect->x=0;
    myRect->y=0;
    myRect->width=10;
    myRect->height=5;
    showRect(*myRect);
}

void showRect(Rectangle theRect){
    printf("X:%i Y:%i width:%i height:%i\n",theRect.x,theRect.y,theRect.width,theRect.height);
}
```

why all of this?

- new way to organize your code
- things that belong together stay together
- you can make arrays of structs
- you can make pointers to structs
- = works on them
- access them using . or if they are pointers ->

good behavior

- naming conventions are a good idea
- usually I use either capital letters for Structures or lowercase t or s leads
- you should have functions wrapping your structures

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

typedef struct rectangle{
    int x,y,width,height;
} Rectangle;

void showRect(Rectangle theRect);
Rectangle makeRect(int x,int y,int width,int height);

int main(int argc,const char* argv[]){
    Rectangle myRect;
    myRect=makeRect(0,0,10,5);
    showRect(myRect);
    printf("%i\n",(int)sizeof(Rectangle));
}

void showRect(Rectangle theRect){
    printf("X:%i Y:%i width:%i height:%i
\n",theRect.x,theRect.y,theRect.width,theRect.height);
}
Rectangle makeRect(int x,int y,int width,int height){
    Rectangle R;
    R.x=x; R.y=y; R.width=width; R.height=height;
    return R;
}
```

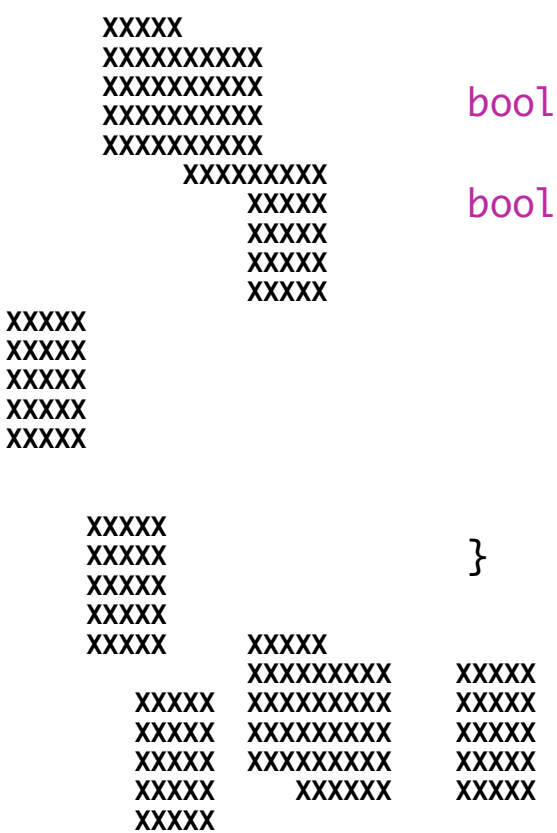
todo:

- get the previous code
- add a function that tests if a point is within a rectangle

```
bool isPointInRect(int X,int Y, Rectangle rect);

bool isPointInRect(int X,int Y, Rectangle rect){
    if((X>=rect.x)&&
       (X<rect.x+rect.width)&&
       (Y>=rect.y)&&
       (Y<rect.y+rect.width))
        return true;
    return false;
}
```

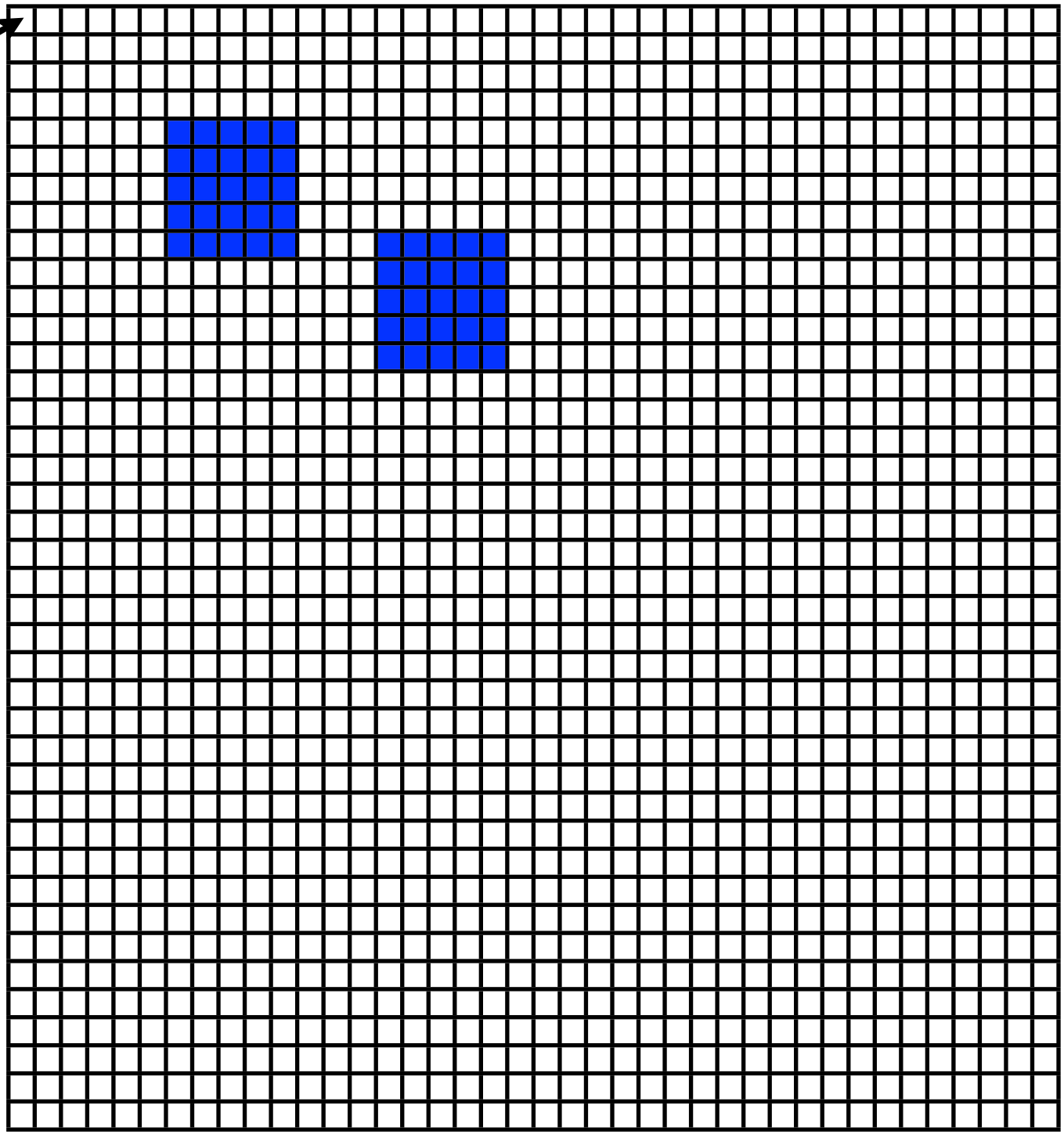
todo:

- make an array of 10 rectangles with random coordinates (0..31,0..31) size 5x5
- draw all of them correctly using X and <space>



```
bool isPointInRect(int X,int Y, Rectangle rect);

bool isPointInRect(int X,int Y, Rectangle rect){
    if((X>=rect.x)&&
       (X<rect.x+rect.width)&&
       (Y>=rect.y)&&
       (Y<rect.y+rect.width))
        return true;
    return false;
}
```

can I set this Y/N



test if point is in any rectangle

```
int main(int argc,const char* argv[]){
    Rectangle rect[10];
    int i,j,k;
    bool b;
    for(i=0;i<10;i++){
        rect[i]=makeRect(rand()&31,rand()&31,5,5);
        showRect(rect[i]);
    }
    for(i=0;i<40;i++){
        for(j=0;j<40;j++){
            b=false;
            for(k=0;k<10;k++)
                if(isPointInRect(i, j, rect[k]))
                    b=true;
            if(b)
                printf("X");
            else
                printf(" ");
        }
        printf("\n");
    }
}
```

union

same as struct, except all variables "overlap" instead of being at different memory locations

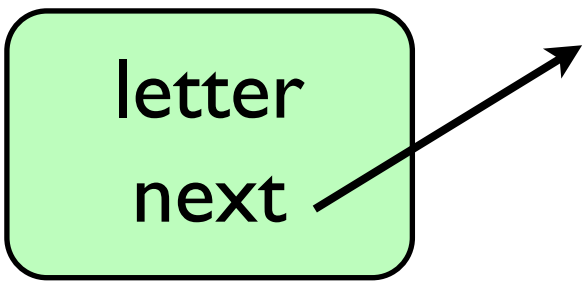
```c
#include <stdio.h>
#include <stdlib.h>

union rectangle{
    int x,y,width,height;
};

void showRect(union rectangle theRect);

int main(int argc,const char* argv[]){
    union rectangle myRect;
    myRect.x=0;
    myRect.y=0;
    myRect.width=10;
    myRect.height=5;
    showRect(myRect);
}

void showRect(union rectangle theRect){
    printf("X:%i Y:%i width:%i height:%i
\n",theRect.x,theRect.y,theRect.width,theRect.height);
}
```

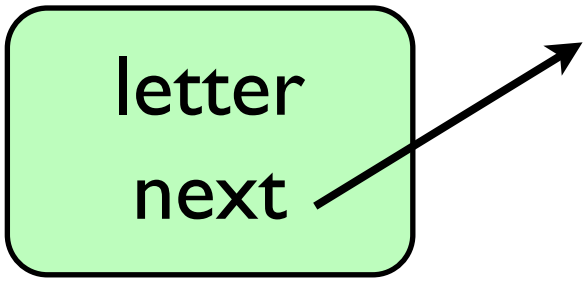
Why unions

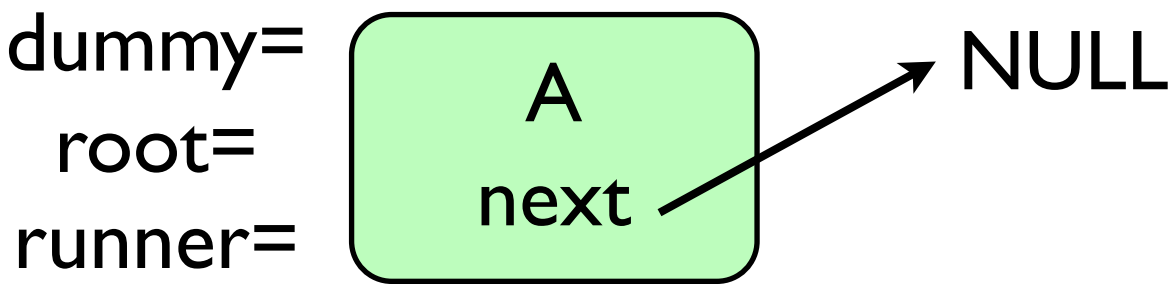
- I never used them
- I asked around ... nobody used them
- I don't know a good example
- Nobody I asked knows a good example
- Hunch: There might be an algorithm that becomes super fast, much better using unions ...

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

typedef struct node{
    char letter;
    struct node *next;
} Node;
```

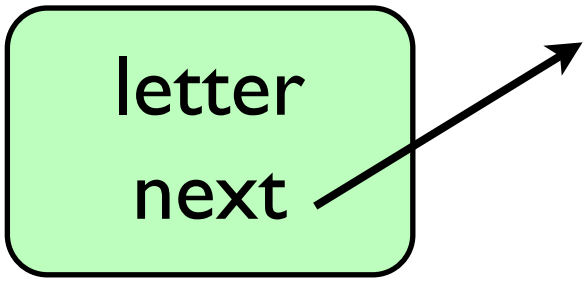
```c
int main(int argc,const char* argv[]){
    FILE *F;
    char C;
    Node *root,*runner,*dummy;
    root=dummy=runner=NULL;
    F=fopen(argv[1],"r");
    while(!feof(F)){
        fscanf(F,"%c",&C);
        dummy=(Node*)malloc(sizeof(Node));
        dummy->letter=C;
        dummy->next=NULL;
        if(root==NULL){
            root=dummy;
            runner=root;
        } else {
            runner->next=dummy;
            runner=dummy;
        }
    }
    runner=root;
    while(runner!=NULL){
        printf("%c",runner->letter);
        runner=runner->next;
    }
    fclose(F);
}
```

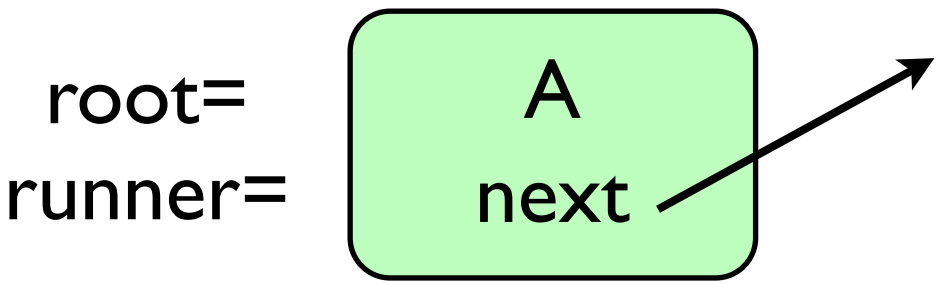

letter
next

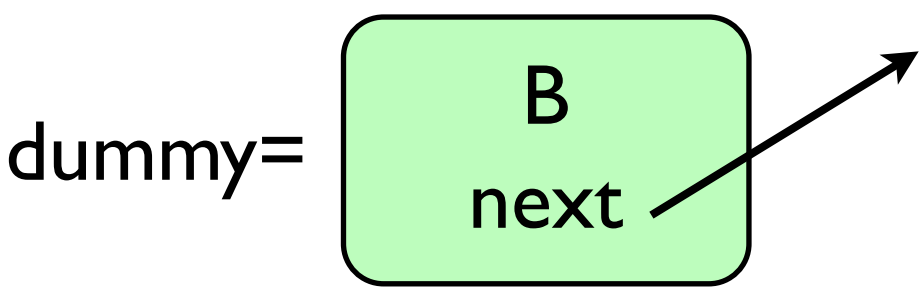




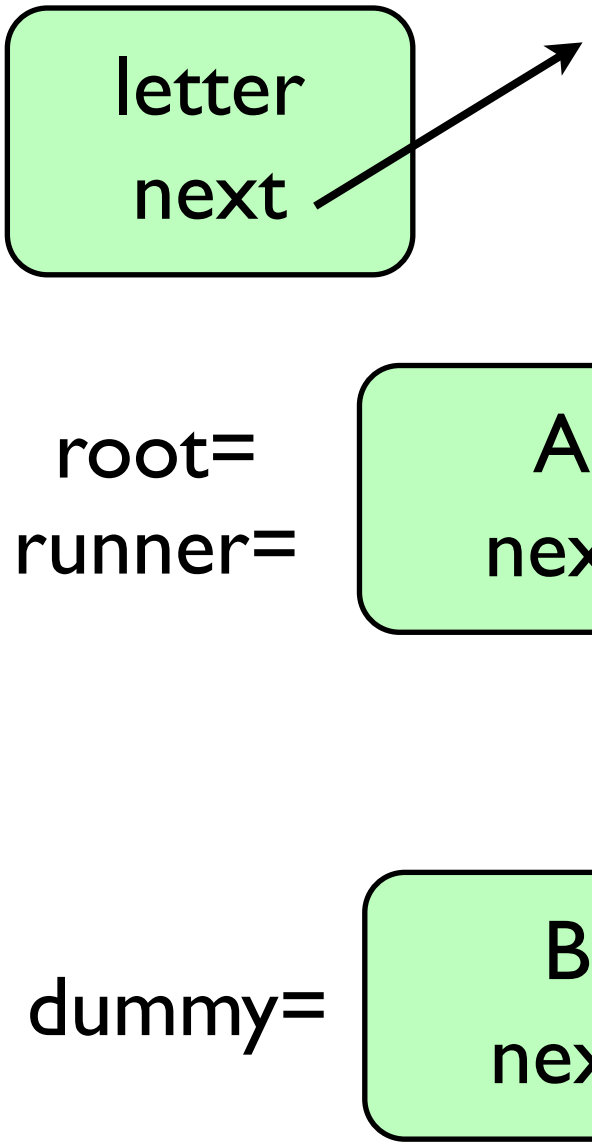

```
while(!feof(F)){
     fscanf(F,"%c",&C);
     dummy=(Node*)malloc(sizeof(Node));
     dummy->letter=C;
     dummy->next=NULL;
     if(root==NULL){
          root=dummy;
          runner=root;
     } else {
          runner->next=dummy;
          runner=dummy;
     }
}
```



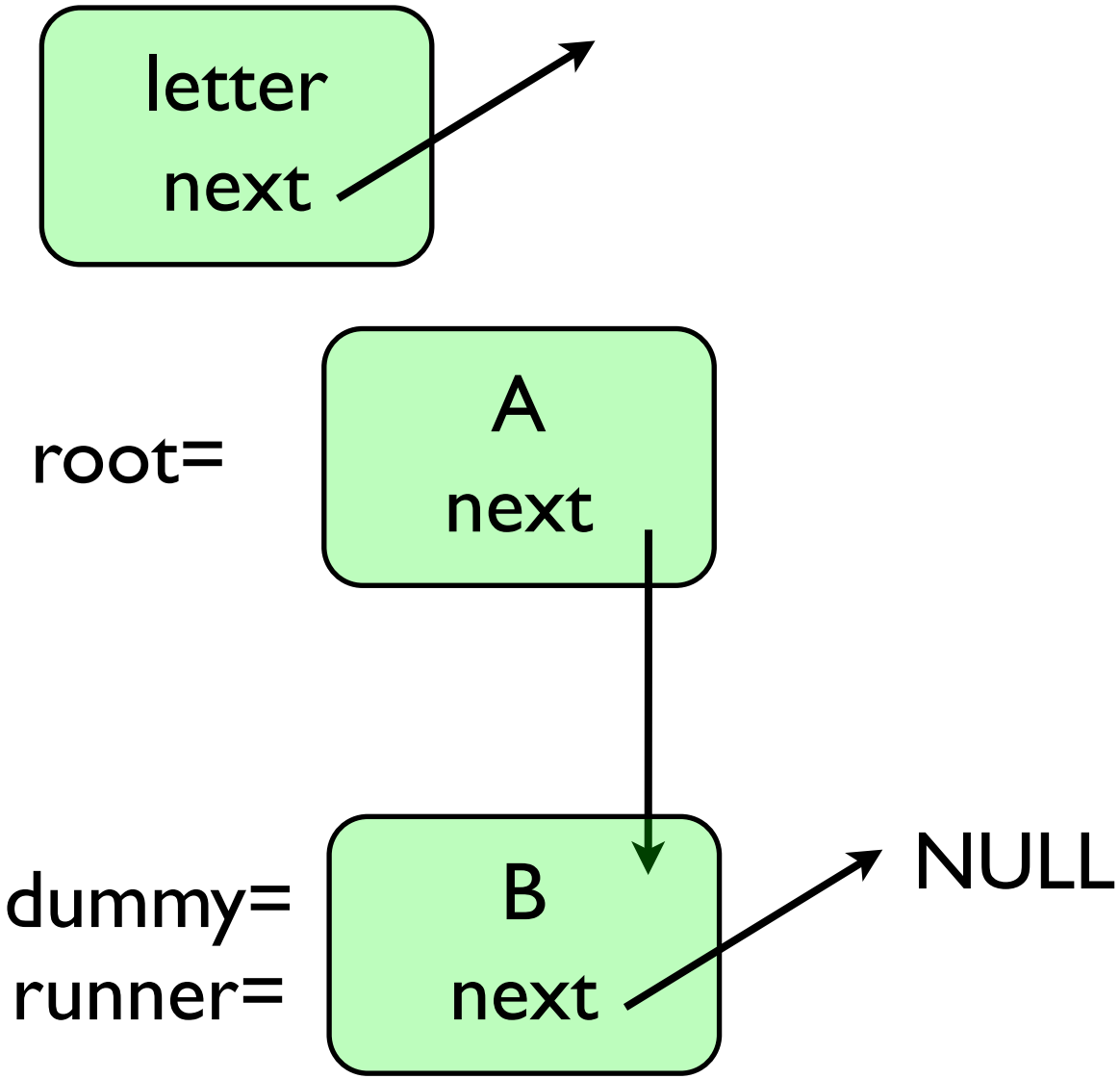




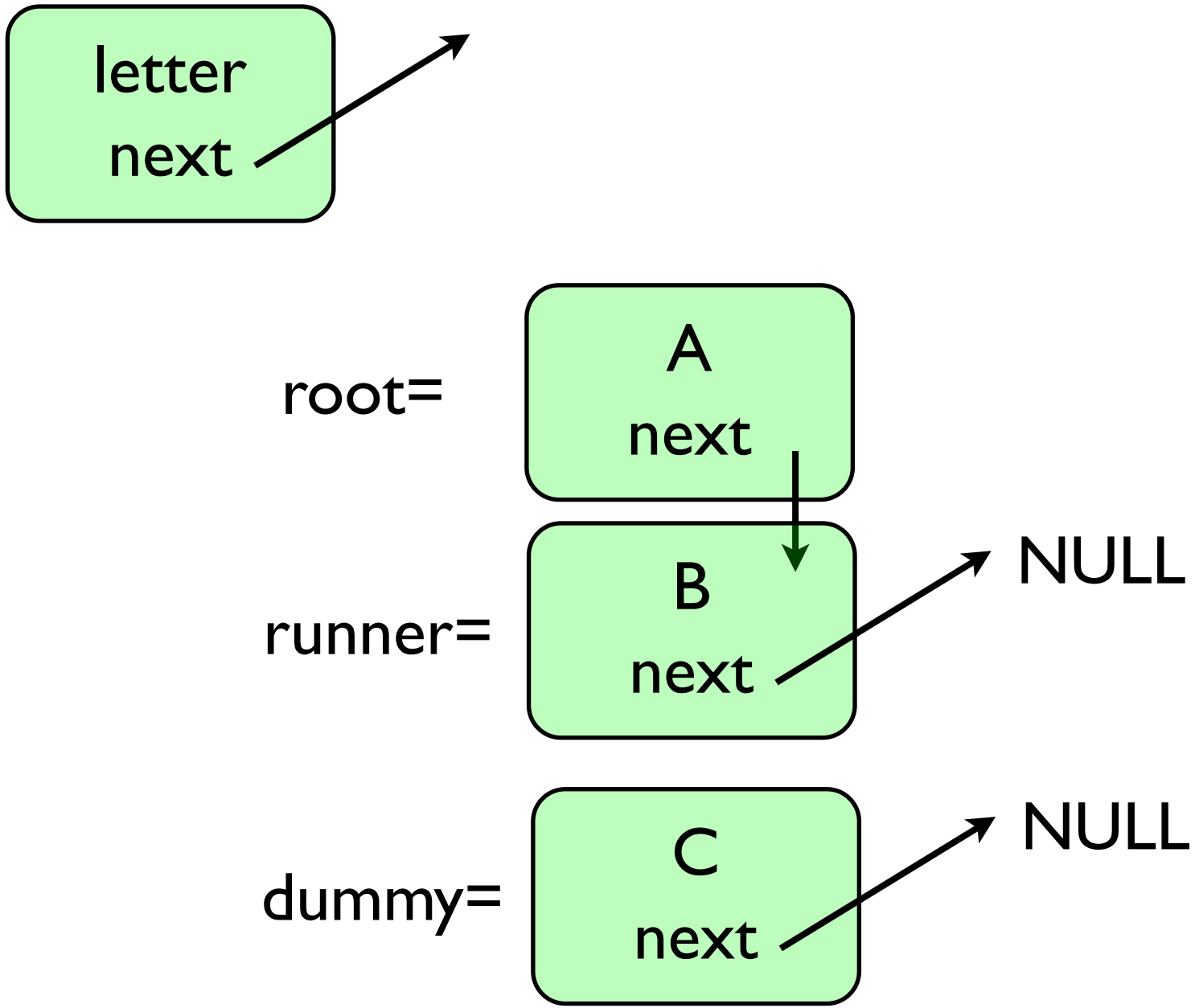

```
while(!feof(F)){
    fscanf(F,"%c",&C);
    dummy=(Node*)malloc(sizeof(Node));
    dummy->letter=C;
    dummy->next=NULL;
    if(root==NULL){
        root=dummy;
        runner=root;
    } else {
        runner->next=dummy;
        runner=dummy;
    }
}
```

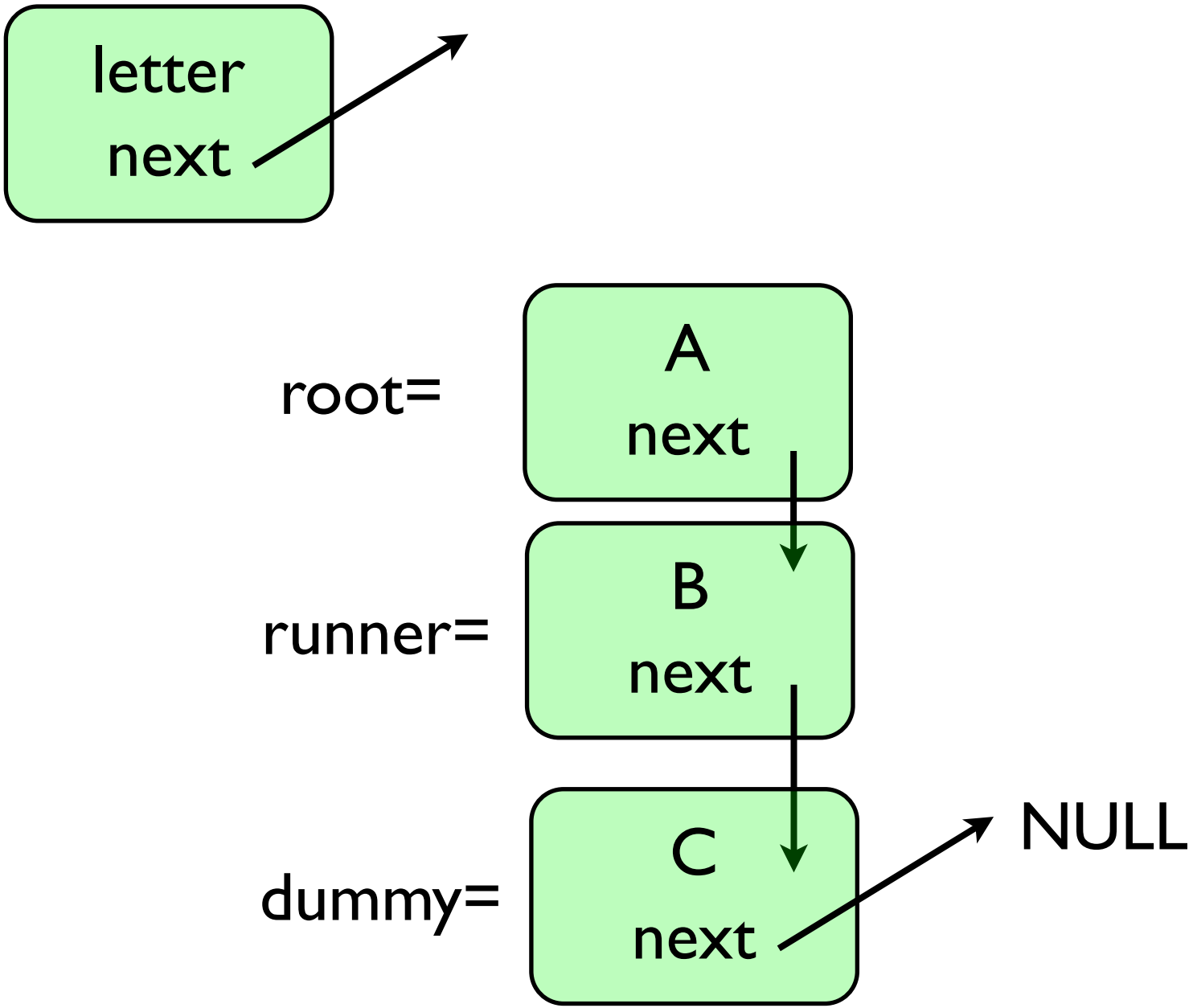


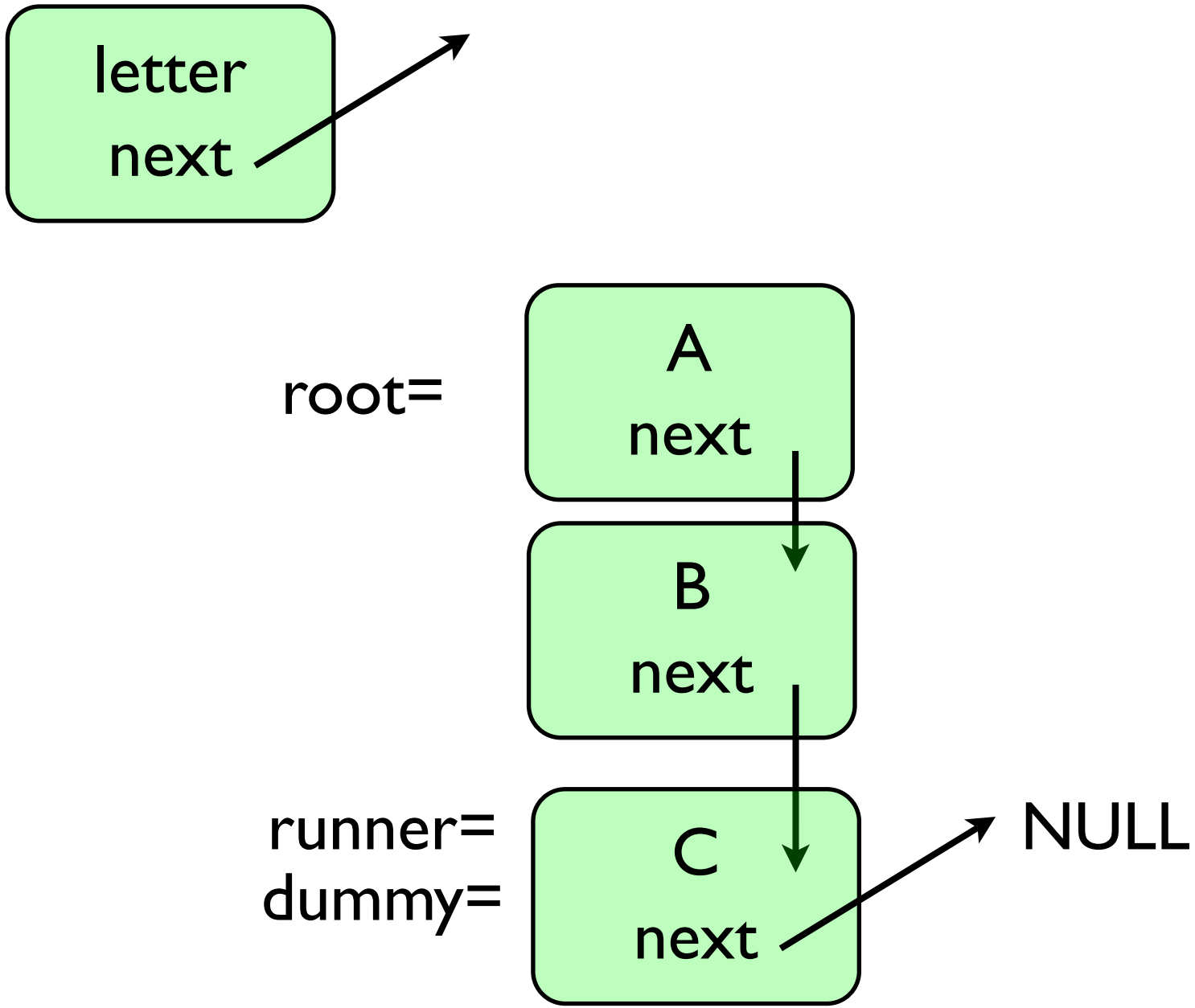

```
while(!feof(F)){
    fscanf(F,"%c",&C);
    dummy=(Node*)malloc(sizeof(Node));
    dummy->letter=C;
    dummy->next=NULL;
    if(root==NULL){
        root=dummy;
        runner=root;
    } else {
        runner->next=dummy;
        runner=dummy;
    }
}
```

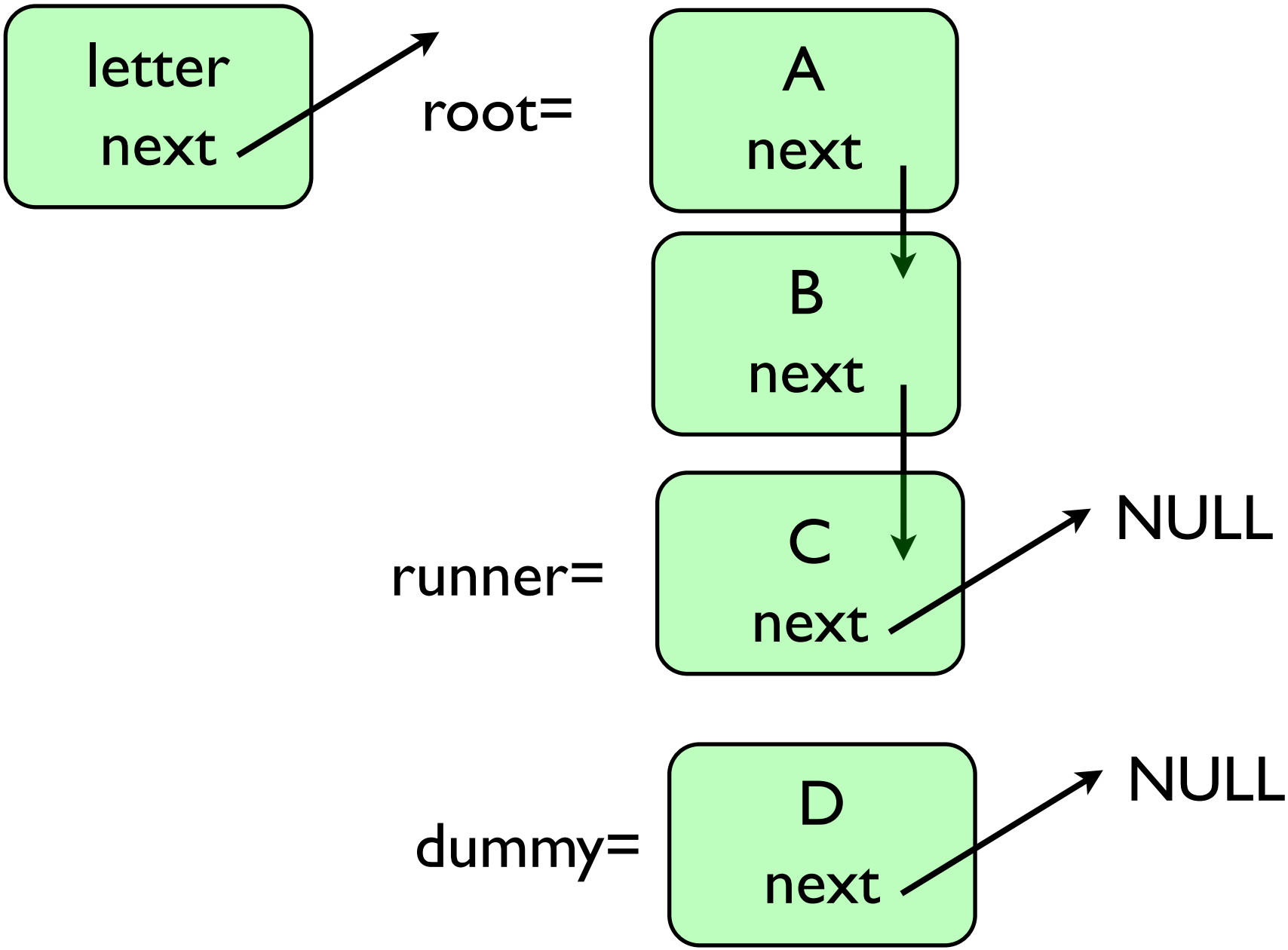



```
while(!feof(F)){
    fscanf(F,"%c",&C);
    dummy=(Node*)malloc(sizeof(Node));
    dummy->letter=C;
    dummy->next=NULL;
    if(root==NULL){
        root=dummy;
        runner=root;
    } else {
        runner->next=dummy;
        runner=dummy;
    }
}
```


letter
next
root=
A
next
runner=
B
next
NULL
dummy=
C
next
NULL


letter
next
root=
A
next
runner=
B
next
dummy=
C
next
NULL


letter
next
root=
A
next
B
next
runner=
dummy=
C
next
NULL


letter
next
root=
A
next
B
next
runner=
C
next
NULL
dummy=
D
next
NULL

homework

- use a linked list to count the word occurrences in a file
- read the file only once, no rewind or fseek!