Pointers and Dynamic Data Structures
Chapter 13

Problem Solving & Program Design in C

Eighth Edition

Jeri R. Hanly & Elliot B. Koffman

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Chapter Objectives

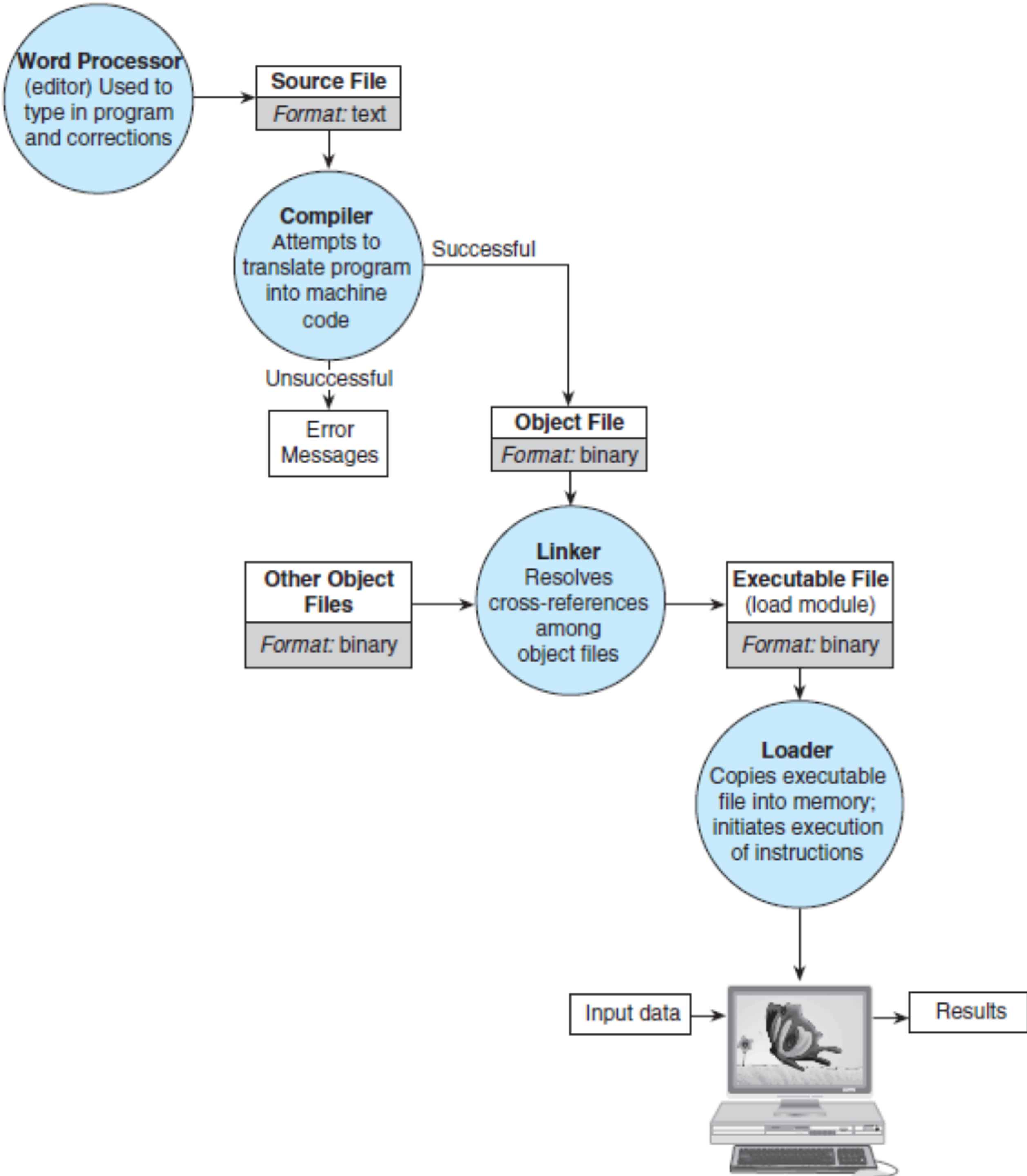
- To understand dynamic allocation on the heap
- To learn how to use pointers to access structs
- To learn how to use pointers to build linked data structures
- To understand how to use and implement a linked list

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Previous uses of pointers…

- Reference to data
- Output parameters
- Arrays and strings
- File pointers

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.


Word Processor (editor) Used to type in program and corrections
Source File
Format: text
Compiler Attempts to translate program into machine code
Successful
Unsuccessful
Error Messages
Object File
Format: binary
Other Object Files
Format: binary
Linker Resolves cross-references among object files
Executable File (load module)
Format: binary
Loader Copies executable file into memory; initiates execution of instructions
Input data
Results

What happens when we run our executable file?

```
func1(int x) {
    x += 1;
    return(x);
}

int main(void) {
    int n = 10;
    n = func1(n);
    return(0);
}
```

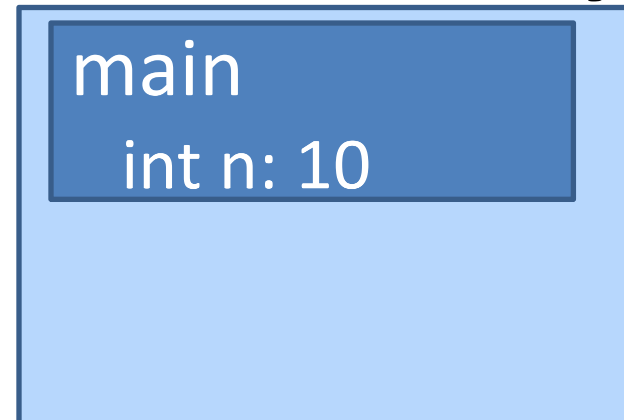
Stack memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
func1(int x) {
   x += 1;
   return(x);
}

int main(void) {
   int n = 10;
   n = func1(n);
   return(0);
}
```

Stack memory

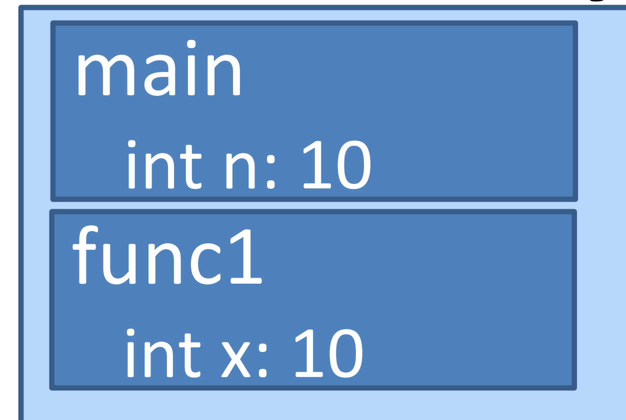


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
func1(int x) {
    x += 1;
    return(x);
}

int main(void) {
    int n = 10;
    n = func1(n);
    return(0);
}
```

Stack memory

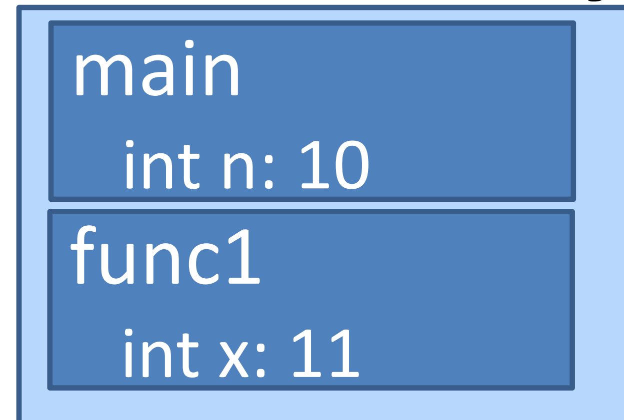


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
func1(int x) {
    x += 1;
    return(x);
}

int main(void) {
    int n = 10;
    n = func1(n);
    return(0);
}
```

Stack memory

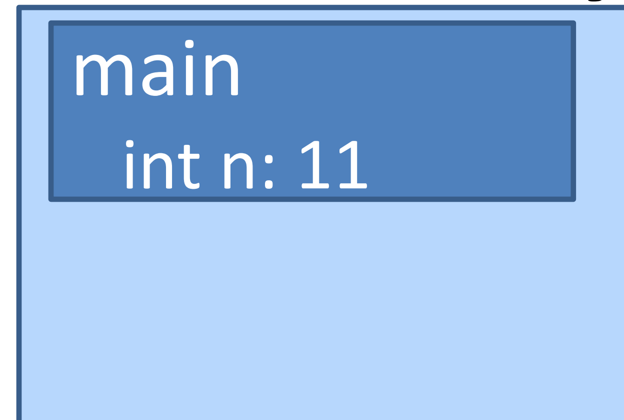


© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
func1(int x) {
    x += 1;
    return(x);
}

int main(void) {
    int n = 10;
    n = func1(n);
    return(0);
}
```

Stack memory



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
func1(int x) {
    x += 1;
    return(x);
}

int main(void) {
    int n = 10;
    n = func1(n);
    return(0);
}
```

Stack memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

Stack memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    free(nump);
}
```

Stack memory

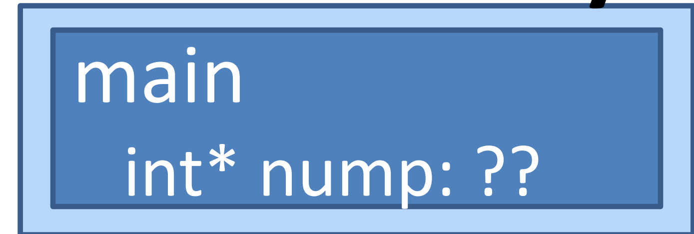
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    free(nump);
}
```

Stack memory



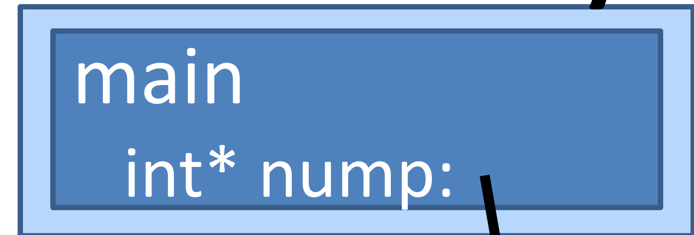
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    free(nump);
}
```

Stack memory



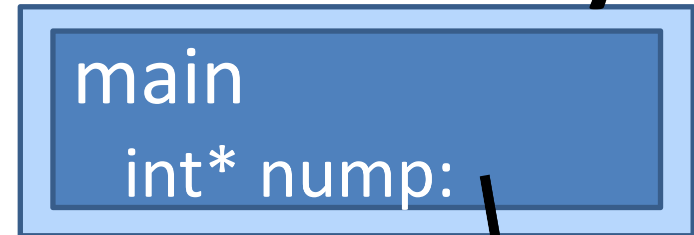
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    free(nump);
}
```

Stack memory



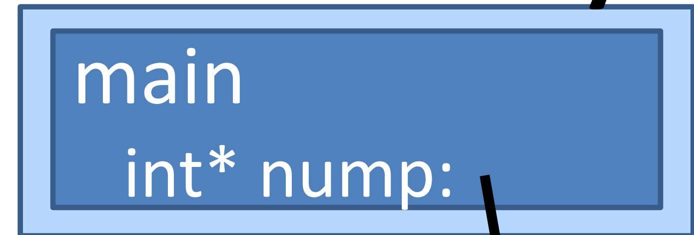
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    free(nump);
}
```

Stack memory



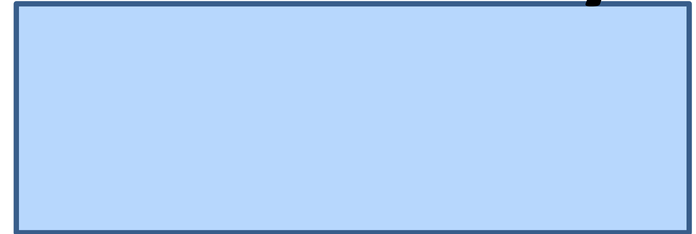
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    free(nump);
}
```

Stack memory



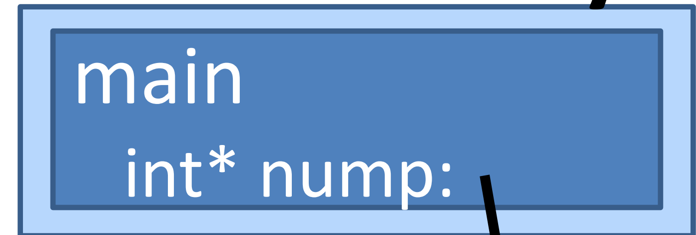
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    free(nump);
    *nump++;
}
```

Stack memory



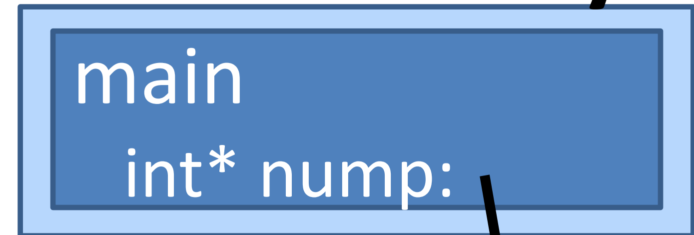
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    free(nump);
    *nump++;
}
```

Stack memory



undefined behavior!

Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Dynamic Memory Allocation

- heap
  - region of memory in which function malloc dynamically allocates blocks of storage

- stack
  - region of memory in which function data areas are allocated and reclaimed

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Important functions

- malloc(<amnt of memory to reserve>)
- calloc(<num>, <amnt of memory to reserve>)
- free(pointer)

These are all from stdlib.h.

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, “hello”);
    free(nump);
}
```

Stack memory

Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, “hello”);
    free(nump);
}
```

Stack memory

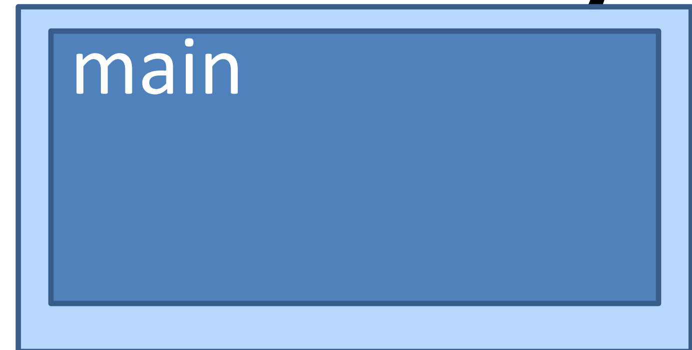
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, “hello”);
    free(nump);
}
```

Stack memory



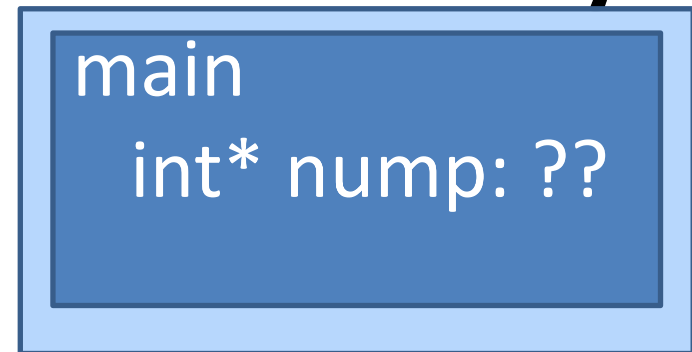
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, "hello");
    free(nump);
}
```

Stack memory



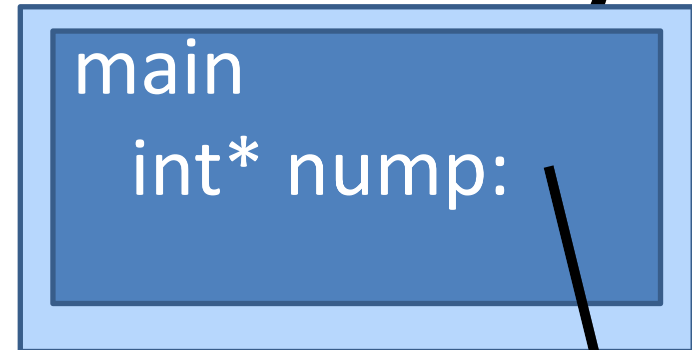
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, "hello");
    free(nump);
}
```

Stack memory



int: ??

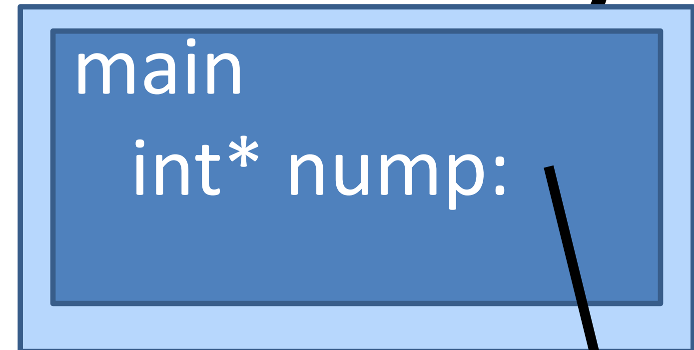
Heap memory

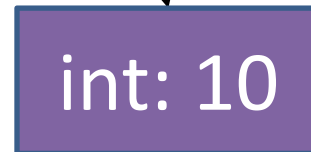
© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, “hello”);
    free(nump);
}
```

Stack memory





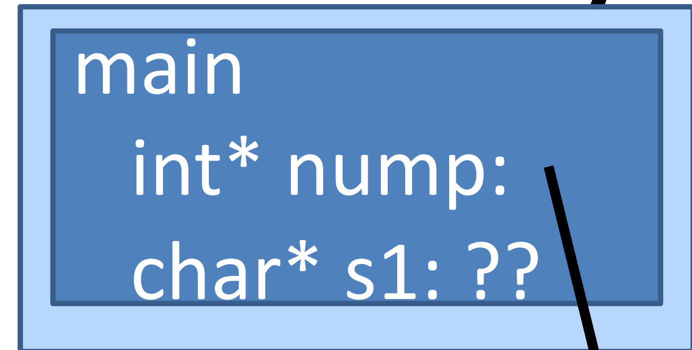
Heap memory

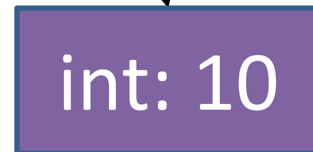
© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, “hello”);
    free(nump);
}
```

Stack memory





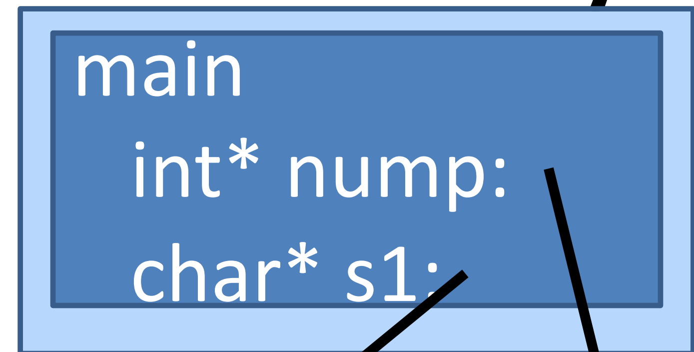
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, "hello");
    free(nump);
}
```

Stack memory



int: 10

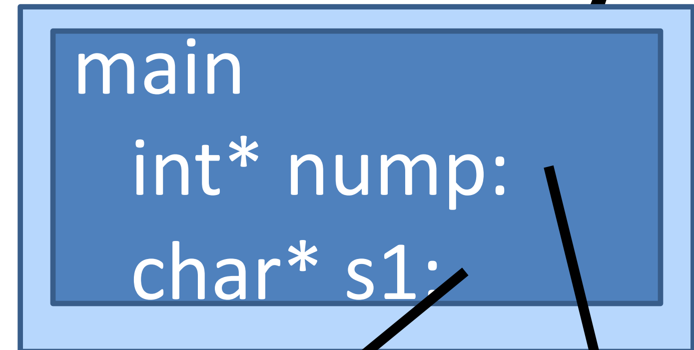
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, “hello”);
    free(nump);
}
```

Stack memory



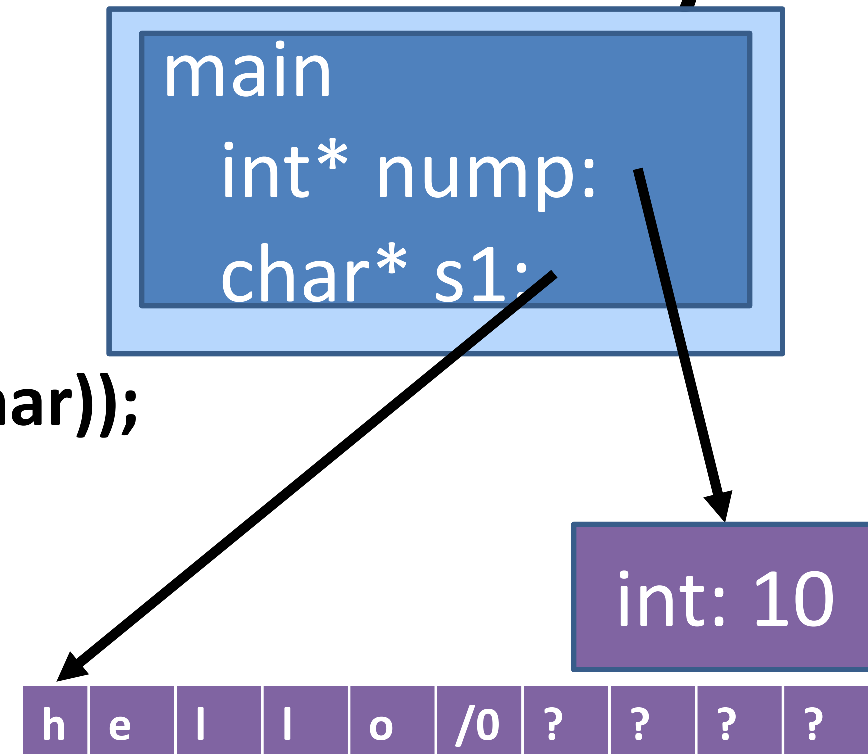
int: 10

Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, “hello”);
    free(nump);
}
```



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, “hello”);
    free(nump);
}
```

Stack memory

h	e	l	l	o	/0	?	?	?	?

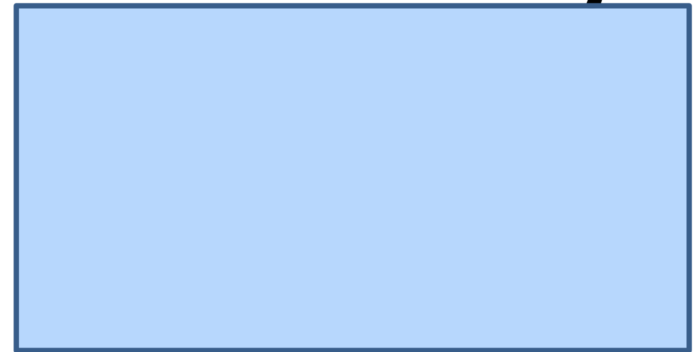
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What happens when we run our executable file?

```
int main(void) {
    int* nump;
    nump = malloc(sizeof(int));
    *nump = 10;
    char* string1;
    string1 = calloc(10, sizeof(char));
    strcpy(string1, “hello”);
    free(nump);
}
```

Stack memory



Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Memory leaks

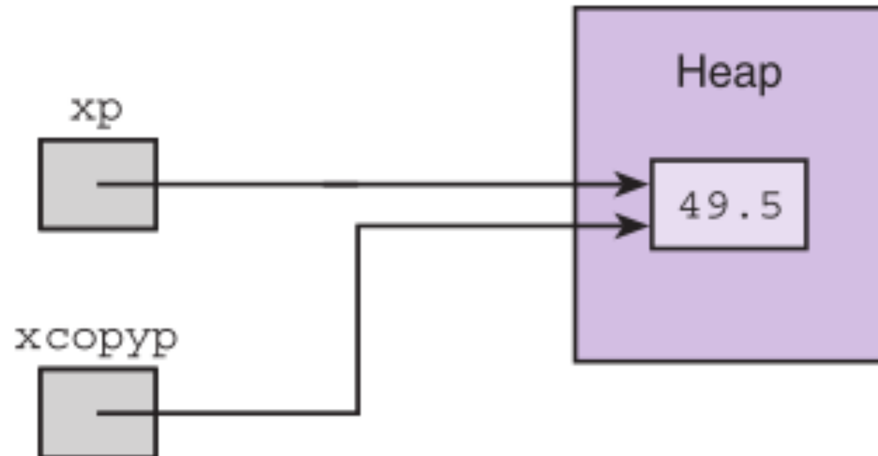
- When not all heap memory is freed before the end of a program
- If time, we'll see a program (valgrind) that can check for memory leaks

(in reality, for a short-running program, not freeing our memory would be okay…but we want to be in the habit of freeing memory!)

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

FIGURE 13.9

Multiple Pointers to a Cell in the Heap




```
double *xp, *xcopyp;

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

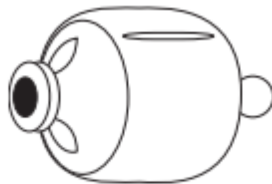
© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

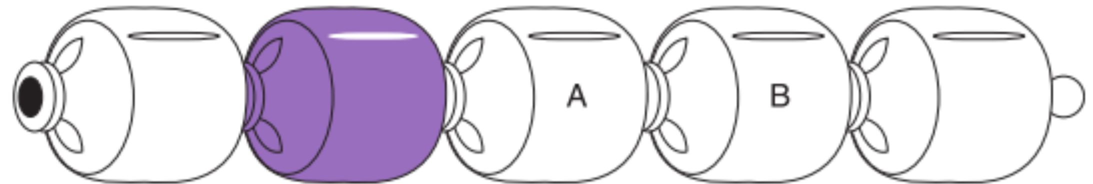
Linked Lists

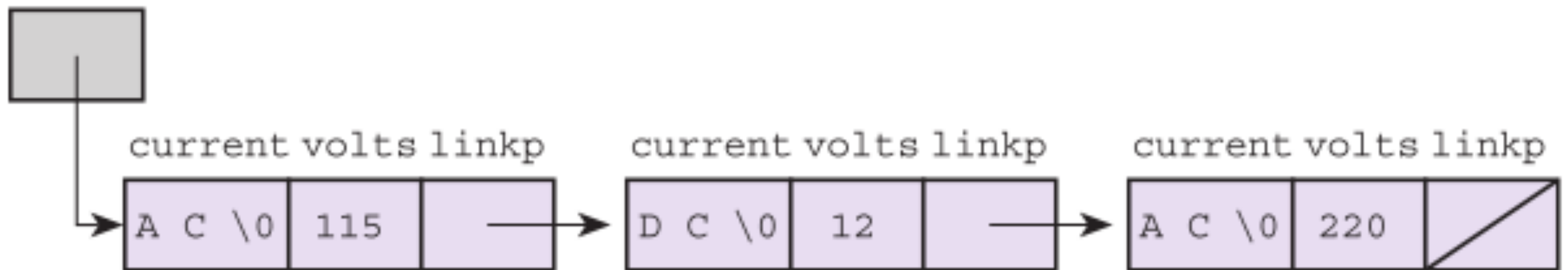
- linked list
  - a sequence of nodes in which each node but the last contains the address of the next node
- empty list
  - a list of no nodes
  - represented in C by the pointer NULL, whose value is zero
- list head
  - the first element in a linked list

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

FIGURE 13.10 Children's Pop Beads in a Chain







© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

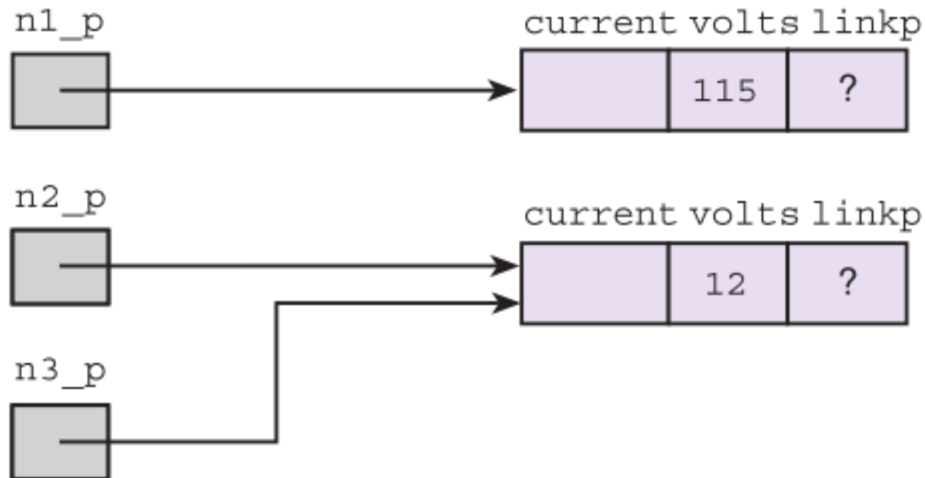


FIGURE 13.11

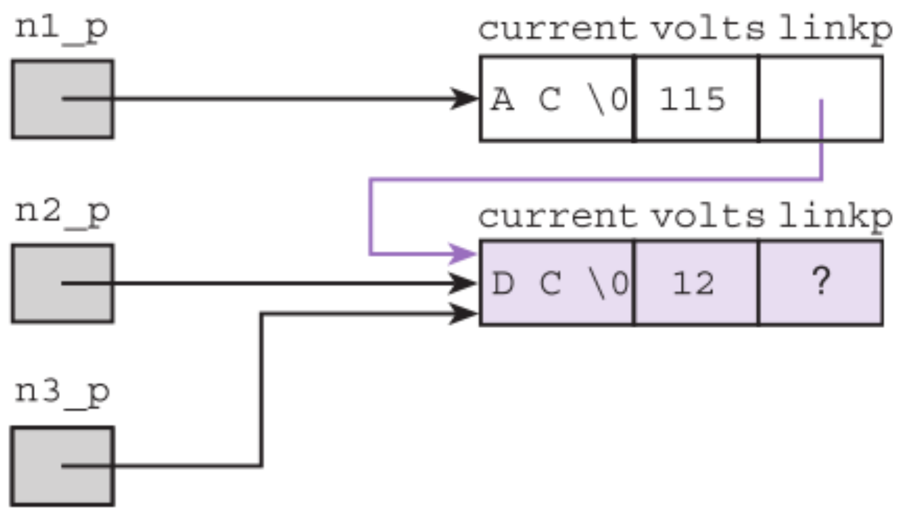
Multiple Pointers to the Same Structure

```
node_t *n1_p, *n2_p, *n3_p;
n1_p = (node_t *)malloc(sizeof (node_t));
strcpy(n1_p->current, "AC");
n1_p->volts = 115;
n2_p = (node_t *)malloc(sizeof (node_t));
strcpy(n2_p->current, "DC");
n2_p->volts = 12;

n3_p = n2_p;
```

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

FIGURE 13.12

Linking Two Nodes



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

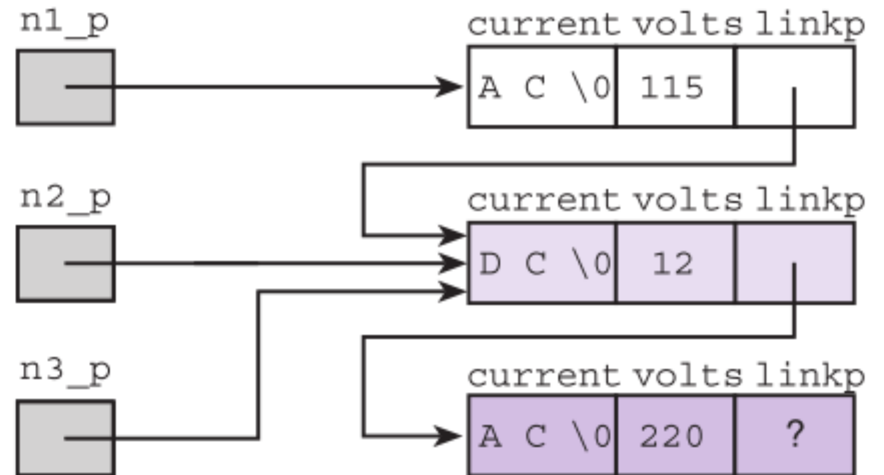
TABLE 13.2 Analyzing the Reference n1_p->linkp->volts

Section of Reference	Meaning
`n1_p->linkp`	Follow the pointer in `n1_p` to a structure and select the `linkp` component.
`linkp->volts`	Follow the pointer in the `linkp` component to another structure and select the `volts` component.

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

FIGURE 13.13

Three-Node Linked List with Undefined Final Pointer



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

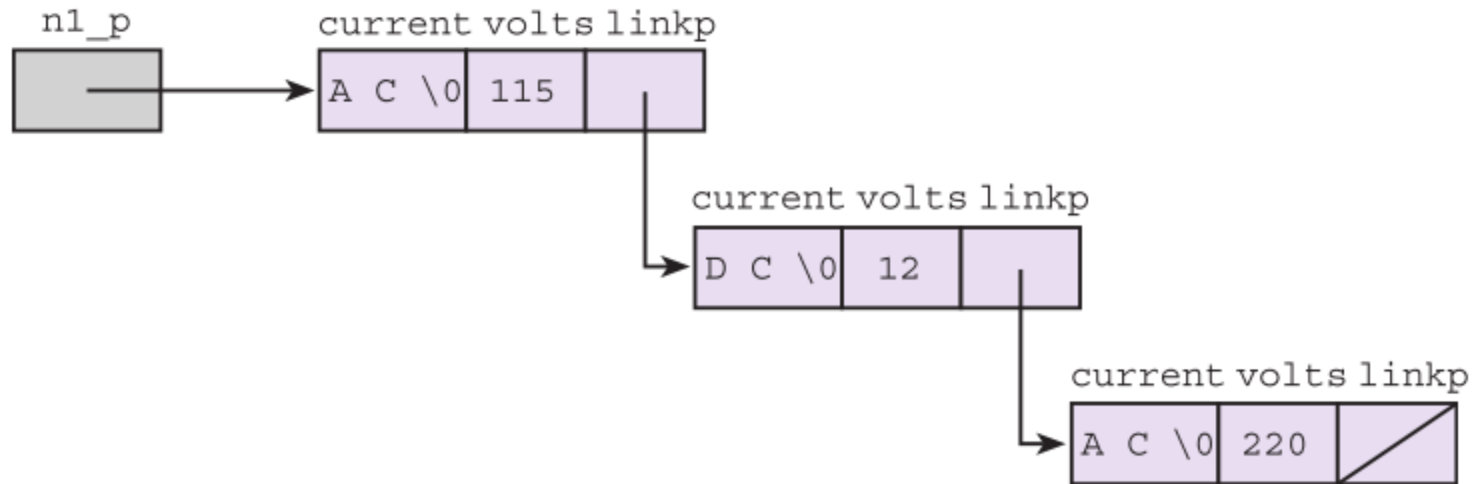


FIGURE 13.14

Three-Element Linked List Accessed Through n1_p

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
digit* create_new_digit(int d) {
   digit* new = malloc(sizeof(digit));
   new->d = d;
   new->next = NULL;
   return(new);
}

int main(void) {
   digit* head;
   head = create_new_digit(1);
   head->next =
         create_new_digit(2);
   head->next->next =
         create_new_digit(3);
}
```

Stack memory

Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```c
digit* create_new_digit(int d) {
   digit* new = malloc(sizeof(digit));
   new->d = d;
   new->next = NULL;
   return(new);
}

int main(void) {
   digit* head;
   head = create_new_digit(1);
   head->next =
         create_new_digit(2);
   head->next->next =
         create_new_digit(3);
}
```

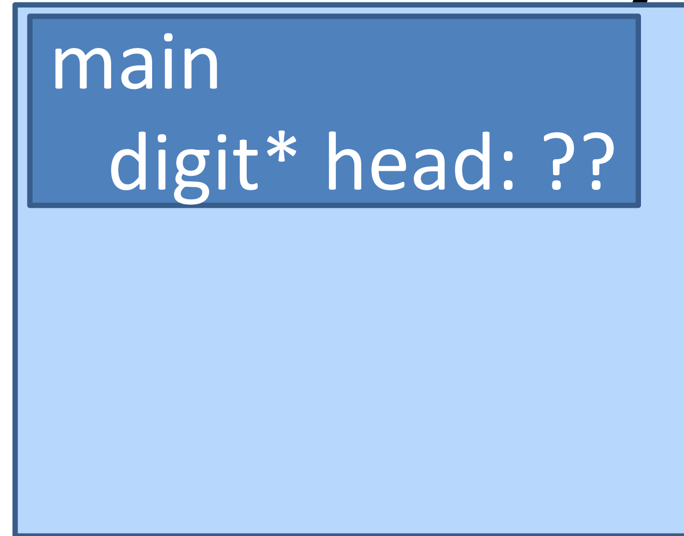
Stack memory

Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
digit* create_new_digit(int d) {
   digit* new = malloc(sizeof(digit));
   new->d = d;
   new->next = NULL;
   return(new);
}

int main(void) {
   digit* head;
   head = create_new_digit(1);
   head->next =
         create_new_digit(2);
   head->next->next =
         create_new_digit(3);
}
```

Stack memory

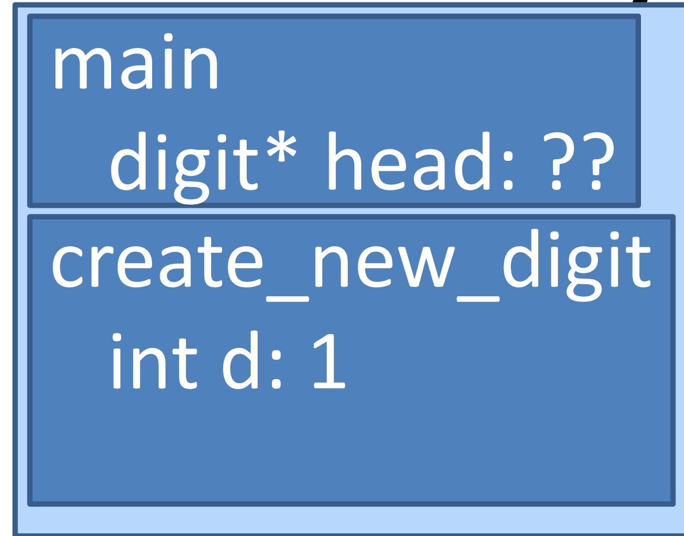


Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
digit* create_new_digit(int d) {
    digit* new = malloc(sizeof(digit));
    new->d = d;
    new->next = NULL;
    return(new);
}

int main(void) {
    digit* head;
    head = create_new_digit(1);
    head->next =
            create_new_digit(2);
    head->next->next =
            create_new_digit(3);
}
```

Stack memory

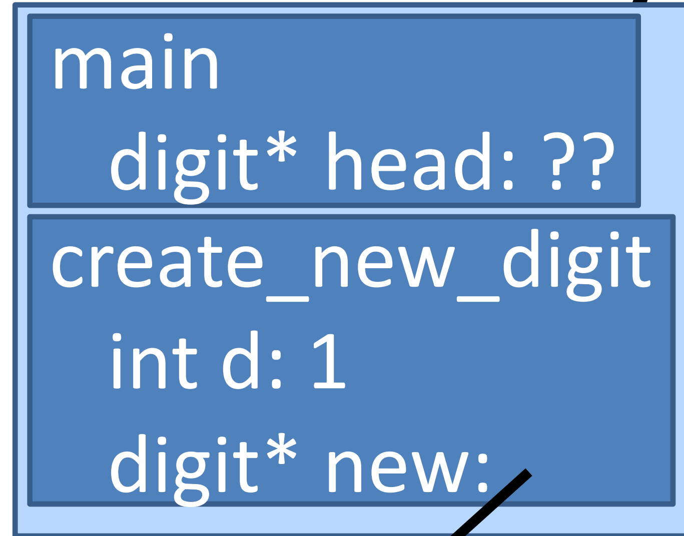


Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
digit* create_new_digit(int d) {
    digit* new = malloc(sizeof(digit));
    new->d = d;
    new->next = NULL;
    return(new);
}

int main(void) {
    digit* head;
    head = create_new_digit(1);
    head->next =
          create_new_digit(2);
    head->next->next =
          create_new_digit(3);
}
```

Stack memory

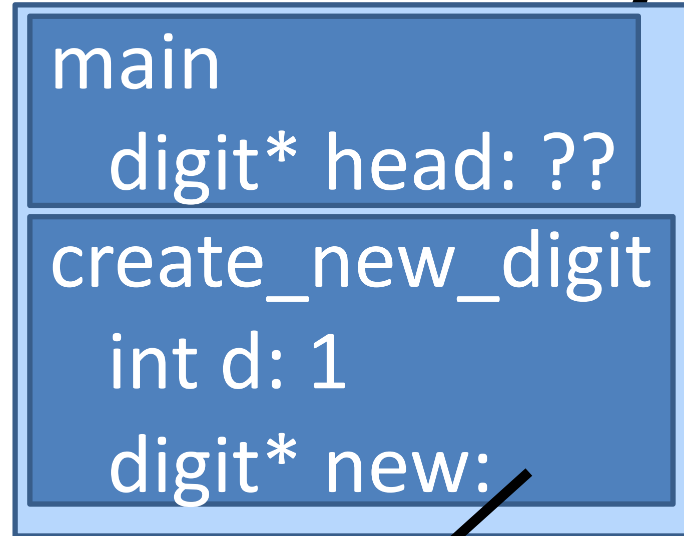


Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
digit* create_new_digit(int d) {
    digit* new = malloc(sizeof(digit));
    new->d = d;
    new->next = NULL;
    return(new);
}

int main(void) {
    digit* head;
    head = create_new_digit(1);
    head->next =
            create_new_digit(2);
    head->next->next =
            create_new_digit(3);
}
```

Stack memory

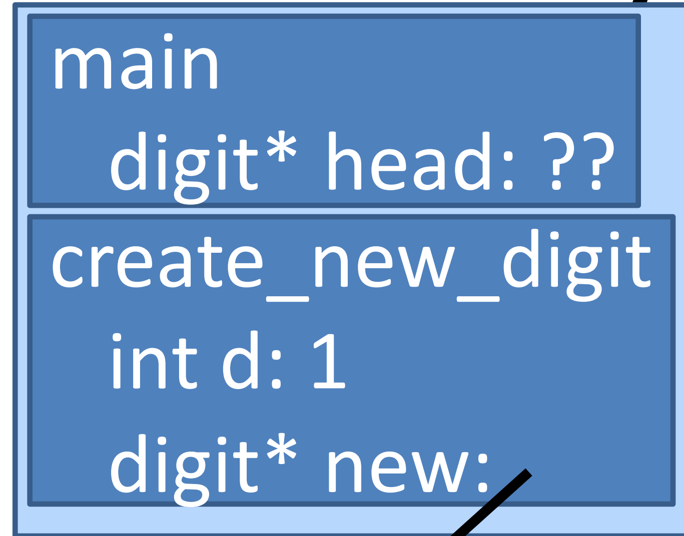


Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
digit* create_new_digit(int d) {
    digit* new = malloc(sizeof(digit));
    new->d = d;
    new->next = NULL;
    return(new);
}

int main(void) {
    digit* head;
    head = create_new_digit(1);
    head->next =
            create_new_digit(2);
    head->next->next =
            create_new_digit(3);
}
```

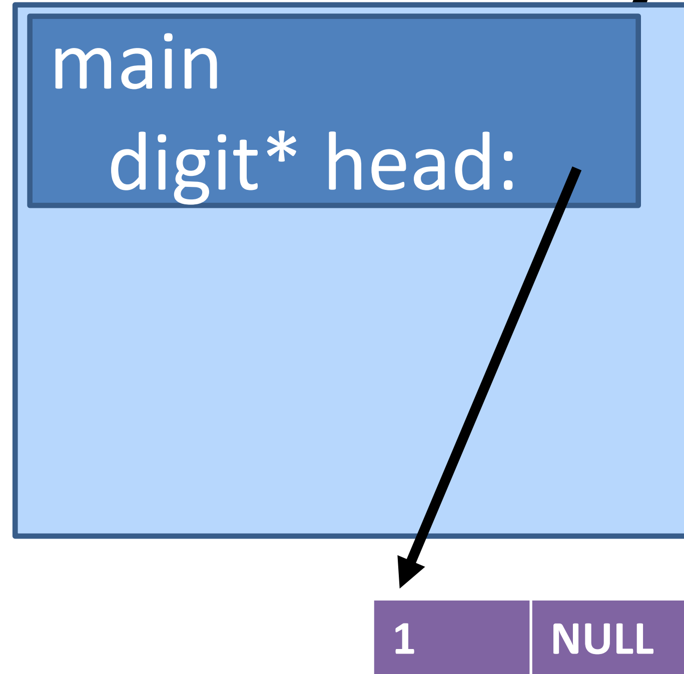
Stack memory



Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

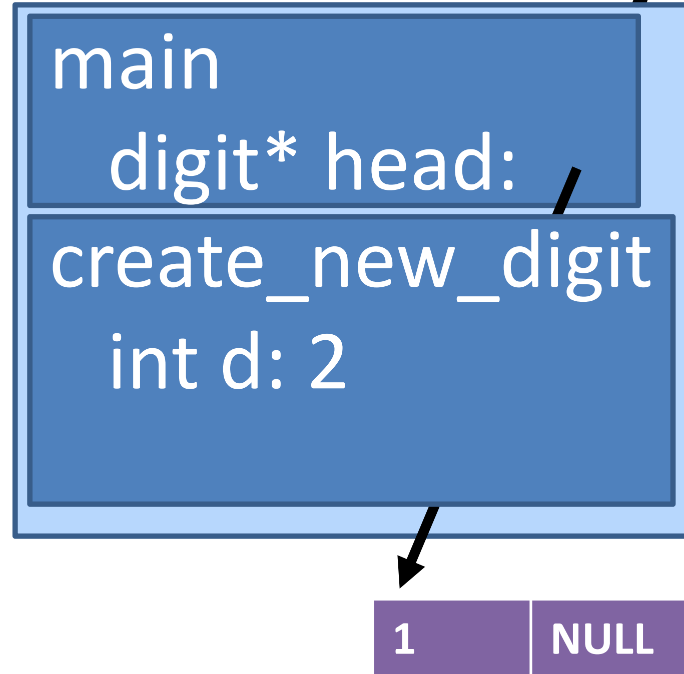
```
digit* create_new_digit(int d) {
   digit* new = malloc(sizeof(digit));
   new->d = d;
   new->next = NULL;
   return(new);
}

int main(void) {
   digit* head;
   head = create_new_digit(1);
   head->next =
         create_new_digit(2);
   head->next->next =
         create_new_digit(3);
}
```



Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

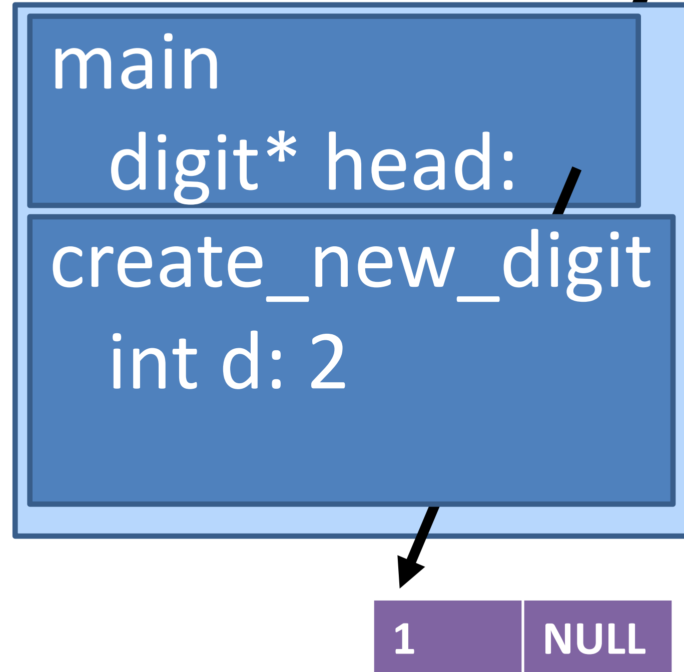
```
digit* create_new_digit(int d) {
    digit* new = malloc(sizeof(digit));
    new->d = d;
    new->next = NULL;
    return(new);
}

int main(void) {
    digit* head;
    head = create_new_digit(1);
    head->next =
            create_new_digit(2);
    head->next->next =
            create_new_digit(3);
}
```



Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
digit* create_new_digit(int d) {
    digit* new = malloc(sizeof(digit));
    new->d = d;
    new->next = NULL;
    return(new);
}

int main(void) {
    digit* head;
    head = create_new_digit(1);
    head->next =
            create_new_digit(2);
    head->next->next =
            create_new_digit(3);
}
```

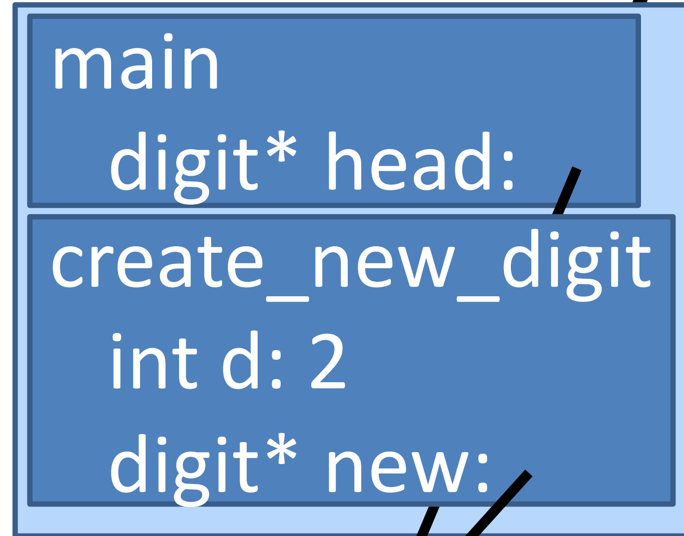


Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
digit* create_new_digit(int d) {
    digit* new = malloc(sizeof(digit));
    new->d = d;
    new->next = NULL;
    return(new);
}

int main(void) {
    digit* head;
    head = create_new_digit(1);
    head->next =
            create_new_digit(2);
    head->next->next =
            create_new_digit(3);
}
```

Stack memory

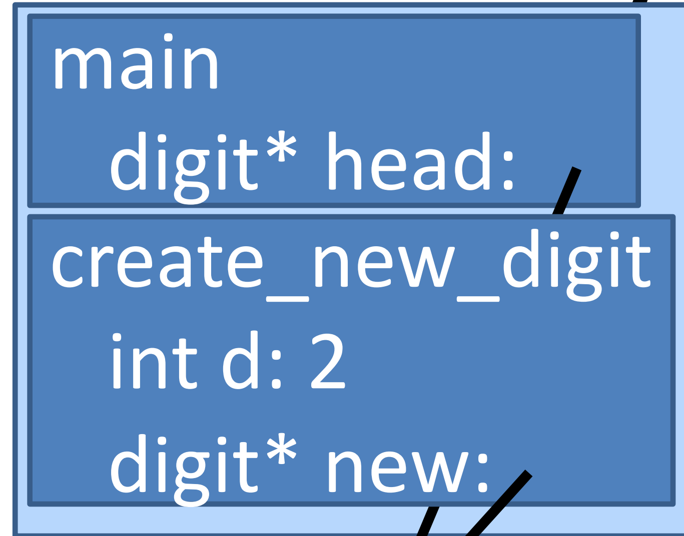


Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
digit* create_new_digit(int d) {
    digit* new = malloc(sizeof(digit));
    new->d = d;
    new->next = NULL;
    return(new);
}

int main(void) {
    digit* head;
    head = create_new_digit(1);
    head->next =
            create_new_digit(2);
    head->next->next =
            create_new_digit(3);
}
```

Stack memory

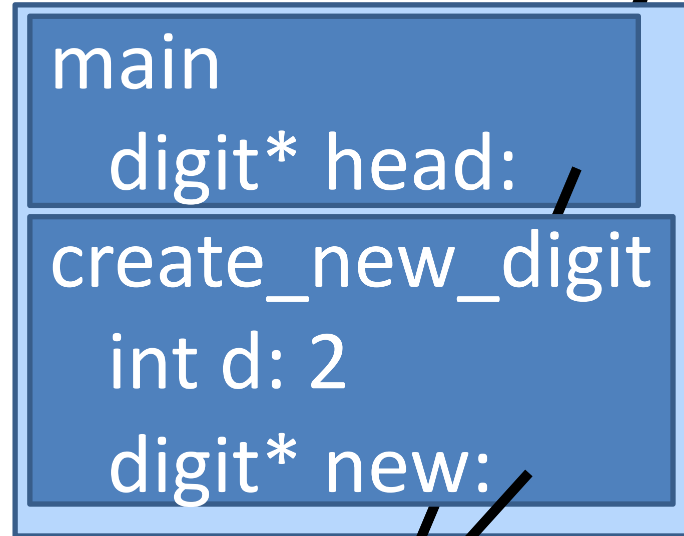


Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
digit* create_new_digit(int d) {
    digit* new = malloc(sizeof(digit));
    new->d = d;
    new->next = NULL;
    return(new);
}

int main(void) {
    digit* head;
    head = create_new_digit(1);
    head->next =
            create_new_digit(2);
    head->next->next =
            create_new_digit(3);
}
```

Stack memory

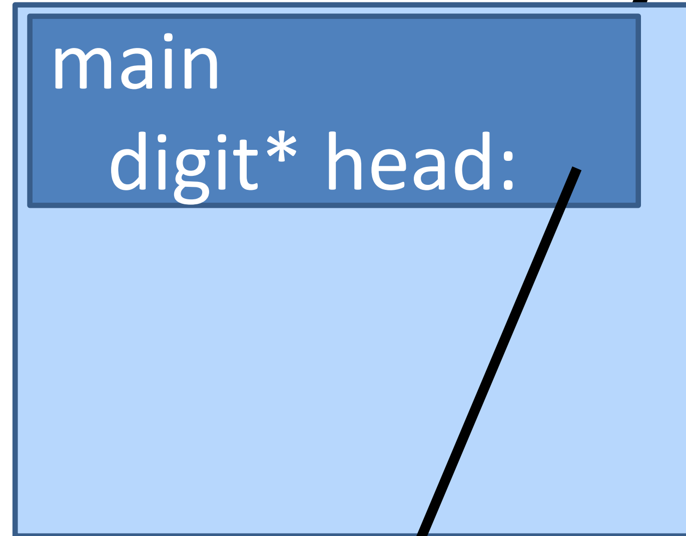


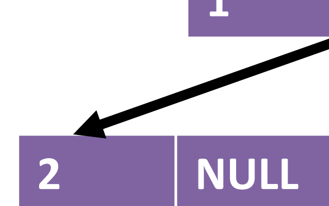
Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

```
digit* create_new_digit(int d) {
    digit* new = malloc(sizeof(digit));
    new->d = d;
    new->next = NULL;
    return(new);
}

int main(void) {
    digit* head;
    head = create_new_digit(1);
    head->next =
            create_new_digit(2);
    head->next->next =
            create_new_digit(3);
}
```

Stack memory

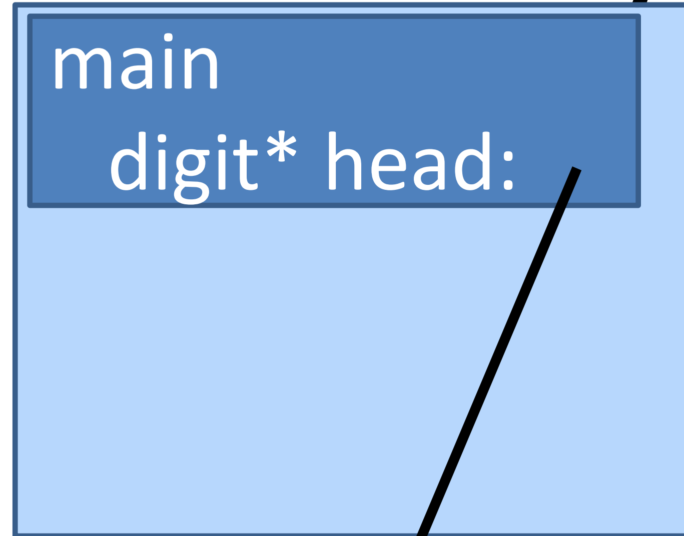


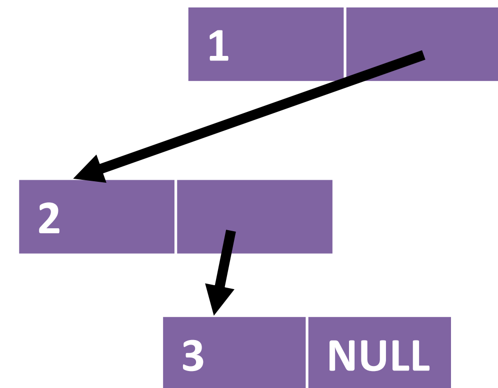


Heap memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

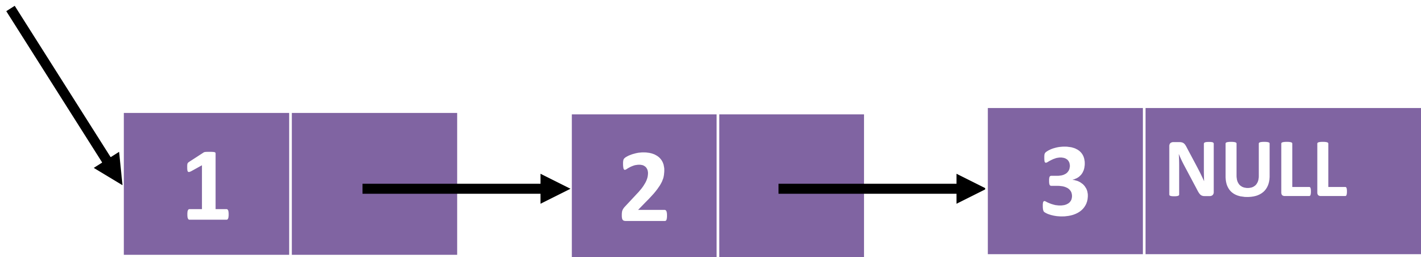
```
digit* create_new_digit(int d) {
   digit* new = malloc(sizeof(digit));
   new->d = d;
   new->next = NULL;
   return(new);
}

int main(void) {
   digit* head;
   head = create_new_digit(1);
   head->next =
          create_new_digit(2);
   head->next->next =
          create_new_digit(3);
}
```





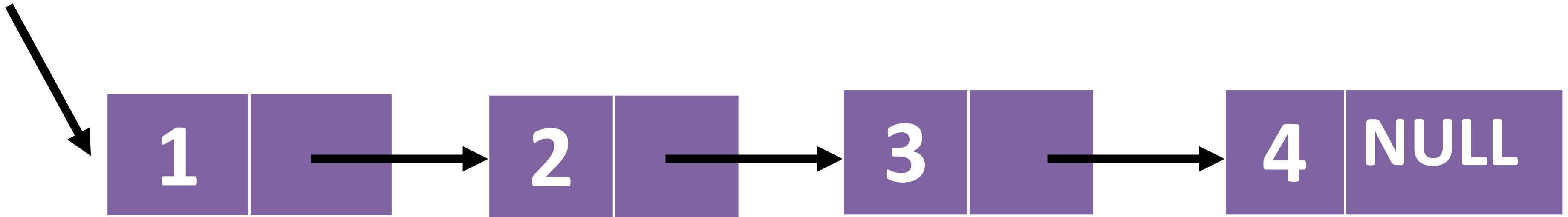
© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Inserting a node at end of list



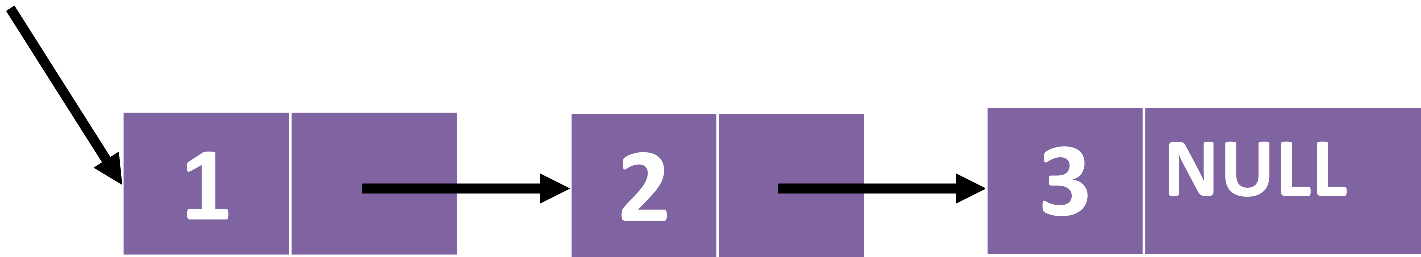
© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Inserting a node at end of list



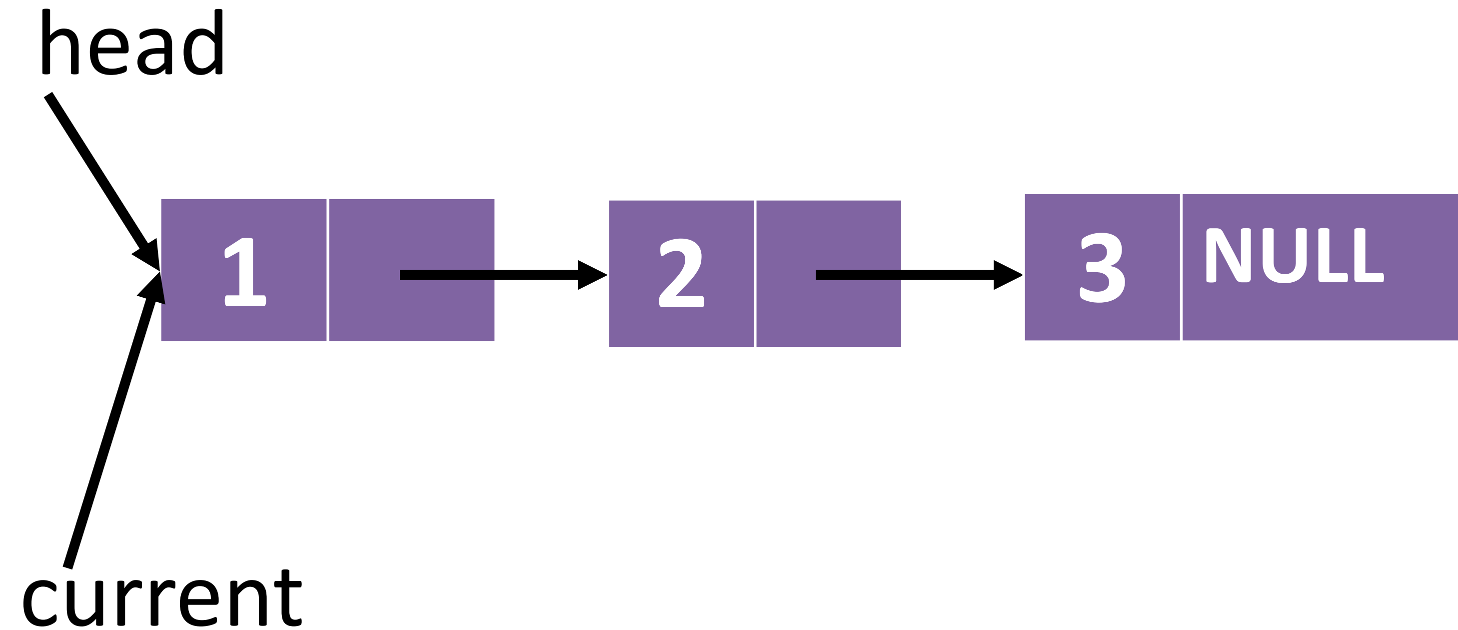
© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Inserting a node at end of list



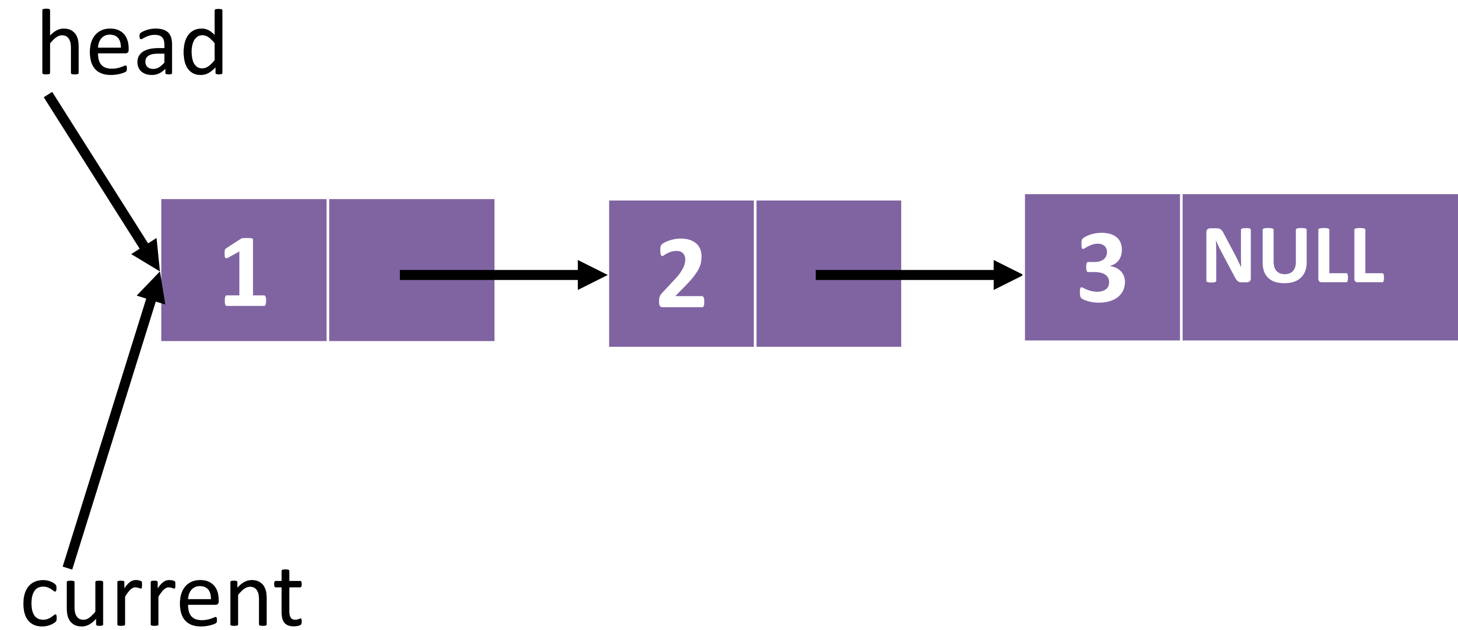
© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Inserting a node at end of list



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

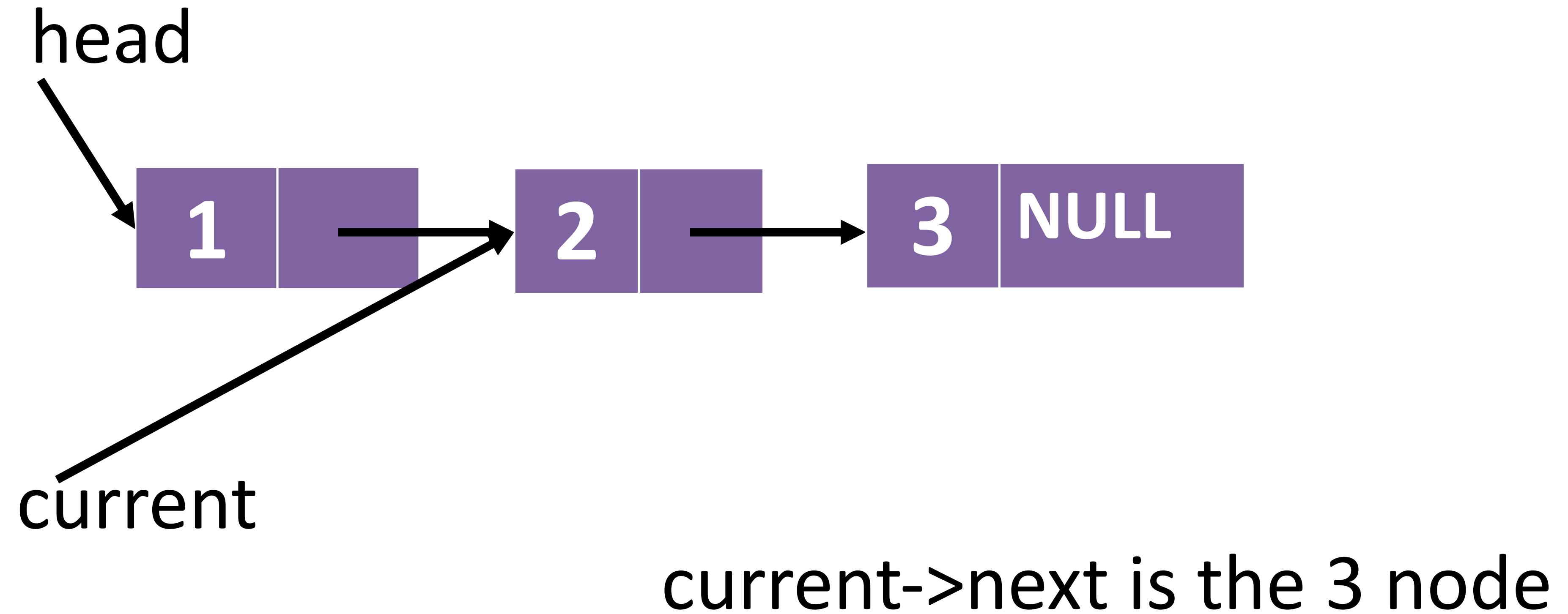
Inserting a node at end of list



current->next is the 2 node

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

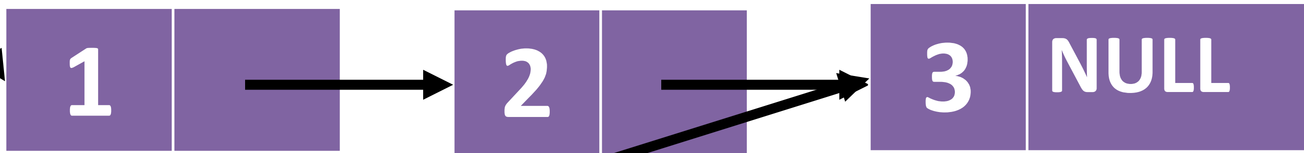
Inserting a node at end of list



current->next is the 3 node

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

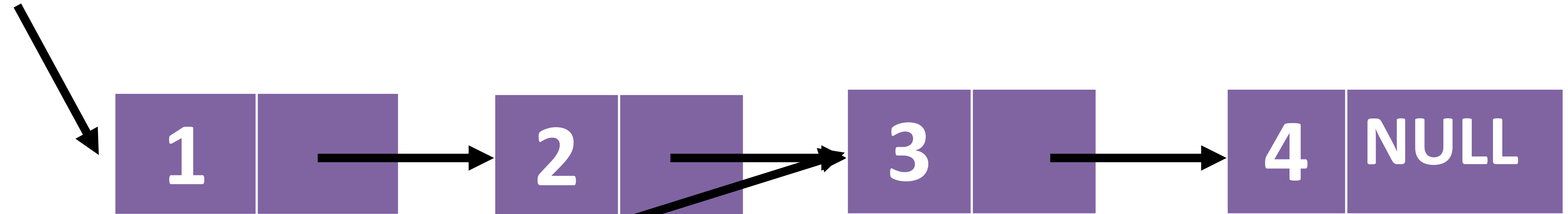
Inserting a node at end of list



current->next == NULL

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

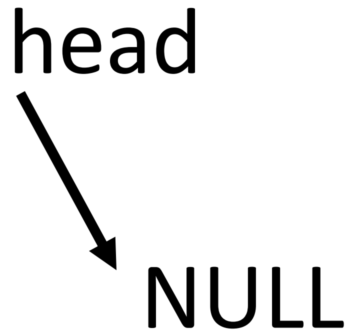
Inserting a node at end of list



current->next = new

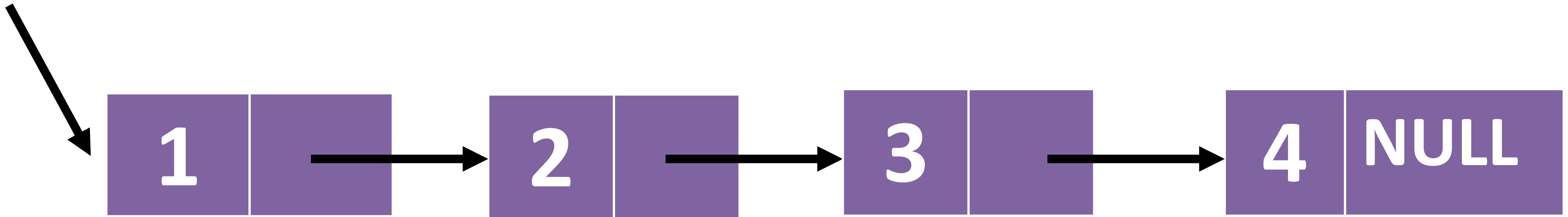
© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

What if list is empty?



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

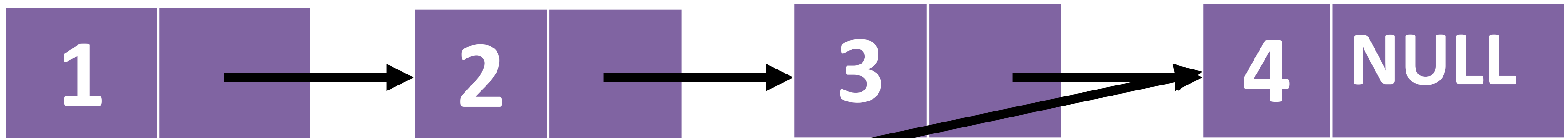
Deleting a node at end of list



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Deleting a node at end of list

head



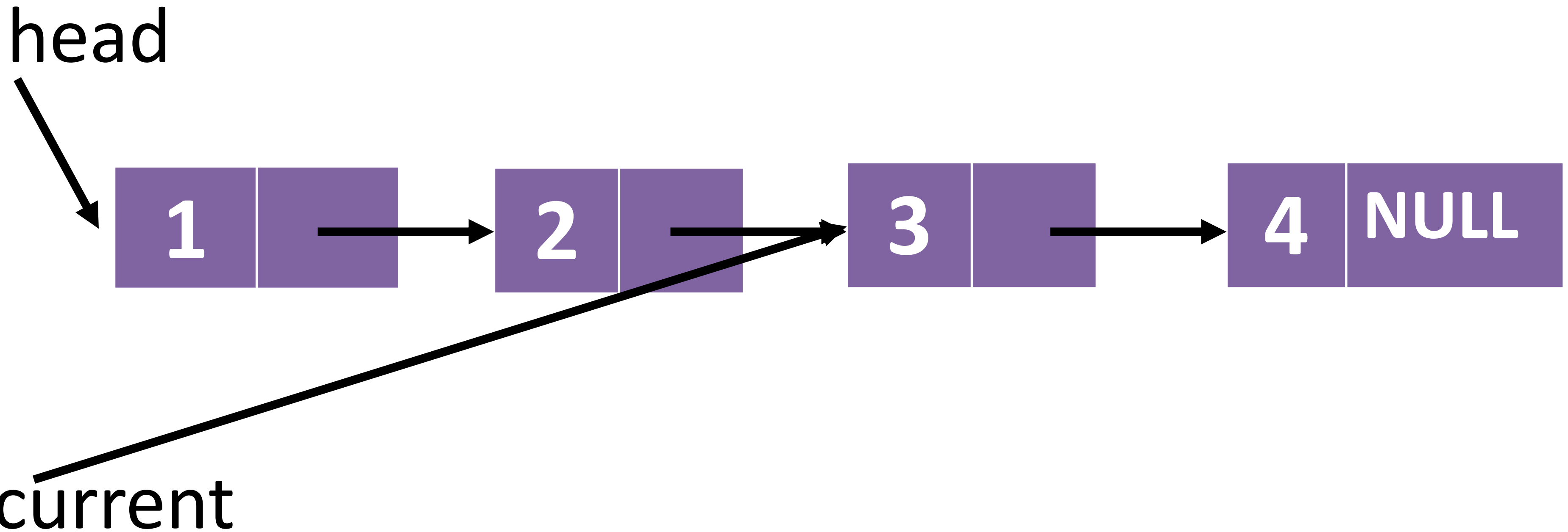
current

current->next == NULL

??? how do we set 3’s next?

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Deleting a node at end of list



current->next->next == NULL

current->next = NULL

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

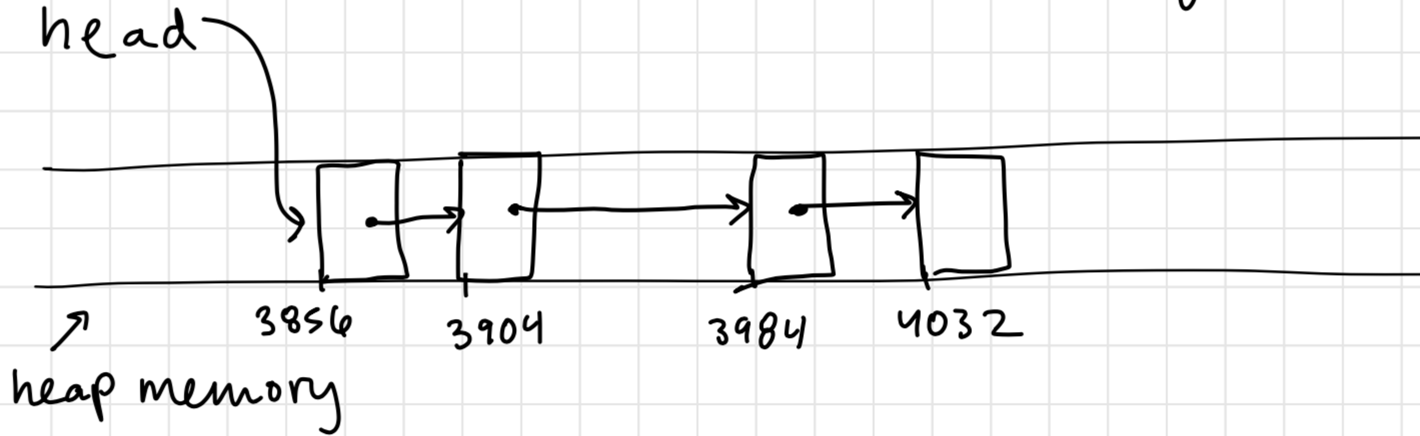
Realloc

- reallocates heap memory
- realloc(ptr, new_size) returns a pointer to a block of memory of new_size with data from ptr copied over, and frees old memory (if needed)
- careful in case realloc is unsuccessful

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Linked List

- not contiguous



Array