Recitation 02

Recursion & Pointers

George Mappouras

9/15/2017

Recursion

- Have a base case were no further computation is needed

- Always make forward progress towards that base case

Recursion

- Have a base case were no further computation is needed
- Always make forward progress towards that base case

```
unsigned long recFactorial(int n)
{
    if (n > 1)
        return n*recFactorial(n-1);
    else
        return 1;
}
```

Types of recursion

- Head recursion

  Computation after returning

- Tail recursion

  No computation left after returning

- Mutual recursion

  Multiple functions calling each other

```
unsigned long recFactorial(int n)
{
   if (n >= 1)
      return n*recFactorial(n-1);
   else
      return 1;
}
```

```
unsigned long recFactorial(int n, int ans)
{
   if (n <=0 )
      return ans;

    return recFactorial(n-1, ans*n);
}
```

Head Recursion

```
unsigned long recFactorial(int n)
{
   if (n >= 1)
      return n*recFactorial(n-1);
   else
      return 1;
}
```

Tail Recursion

```
unsigned long recFactorial(int n, int ans)
{
   if (n <=0 )
      return ans;

    return recFactorial(n-1, ans*n);
}
```

Fibonacci

$F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2)$

Fibonacci

F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2)

```
int fib(n){
    if (n<2) then return n
    else return F(n-1) + F(n-2)
}
```

Lets find a better way to implement Fibonacci

Observation

$F(n) = 1*F(n-1) + 1*F(n-2)$

$F(n-1) = 1*F(n-1) + 0*F(n-2)$

Lets find a better way to implement Fibonacci

Observation

F(n) = 1*F(n-1) + 1*F(n-2)
F(n-1) = 1*F(n-1) + 0*F(n-2)

$$\rightarrow \begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} F(n-1) \\ F(n-2) \end{bmatrix}$$

Lets find a better way to implement Fibonacci

Observation

F(n) = 1*F(n-1) + 1*F(n-2)
F(n-1) = 1*F(n-1) + 0*F(n-2)

$$\rightarrow \begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} F(n-1) \\ F(n-2) \end{bmatrix}$$

with the same reasoning: $$\begin{bmatrix} F(n-1) \\ F(n-2) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} F(n-2) \\ F(n-3) \end{bmatrix}$$

Lets find a better way to implement Fibonacci

Observation

F(n) = 1*F(n-1) + 1*F(n-2)
F(n-1) = 1*F(n-1) + 0*F(n-2)

$$\rightarrow \begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} F(n-1) \\ F(n-2) \end{bmatrix}$$

with the same reasoning:

$$\begin{bmatrix} F(n-1) \\ F(n-2) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} F(n-2) \\ F(n-3) \end{bmatrix}$$

Lets find a better way to implement Fibonacci

Observation

n-1 times

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \cdots \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \right) \begin{bmatrix} F(0) \\ F(1) \end{bmatrix}$$

Lets find a better way to implement Fibonacci

So we can conclude

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Question 7.6 : Write a recursive function `unsigned power(unsigned x, unsigned y)` which computes x^y. Write a `main` function which tests it with several values and convince yourself it is correct.

```c
void g(int x) {
  if (x != 0) {
    g(x/10);
    printf("%d", x%10);
  }
}

void f(int x) {
  if (x < 0) {
    printf("-");
    g(-x);
  }
  else if (x == 0) {
    printf("0");
  }
  else {
    g(x);
  }
  printf("\n");
}
```

```c
int main(void) {
  f(42);
  f(-913);
  return EXIT_SUCCESS;
}
```

Question 7.6 : Write a recursive function `unsigned power(unsigned x, unsigned y)` which computes x^y. Write a `main` function which tests it with several values and convince yourself it is correct.

```
#include <stdio.h>
unsigned power(unsigned x, unsigned y){
    if (y==0)
        return 1;
    else if (y==1)
        return x;
    else
        return x*power(x,y-1);
}

int main(){
   unsigned result = power(2,3);
   printf("Result = %u\n",result);
   return 0;
}
```

What is a pointer?

- It’s like a “type” that stores addresses

- We need to define where the address points (in what type of data)

What is a pointer?

- It’s like a “type” that stores addresses

- We need to define where the address points (in what type of data)

Example:

```
int *n;
float *f;
char *c;
```

The * vs &

- The symbol * next to a variable means:

Take the context of the variable, use it as an address, and lead me to the memory with that address

- The symbol & next to a variable means:

Tell me the address of the memory location that stores this variable

Example 1

```
int x = 5;
int *p1 = *x;
int *p2 = &x;
```

Which of the two pointers refers to x?

Answer:

Example 1

```
int x = 5;
int *p1 = *x;
int *p2 = &x;
```

Which of the two pointers refers to x?

Answer: p2

What is an address? Why is it needed

- It is used to refer to places in memory (or nowhere…NULL pointers)
- Used to pass a variable to a function by reference! (and other reasons)

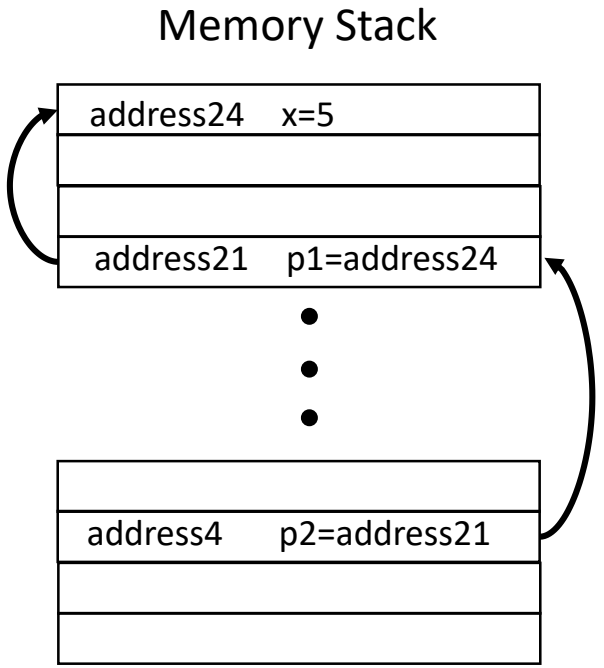
Example:

```
void swap1 (int x, int y){
        int temp = x;
        x=y;
        y=temp;
}
```

```
void swap2 (int *x, int *y){
        int temp = *x;
        *x=*y;
        *y=temp;
}
```

Pointer of a pointer?

```
int x = 5;
int *p1 = &x;
int **p2 = &p1;
```

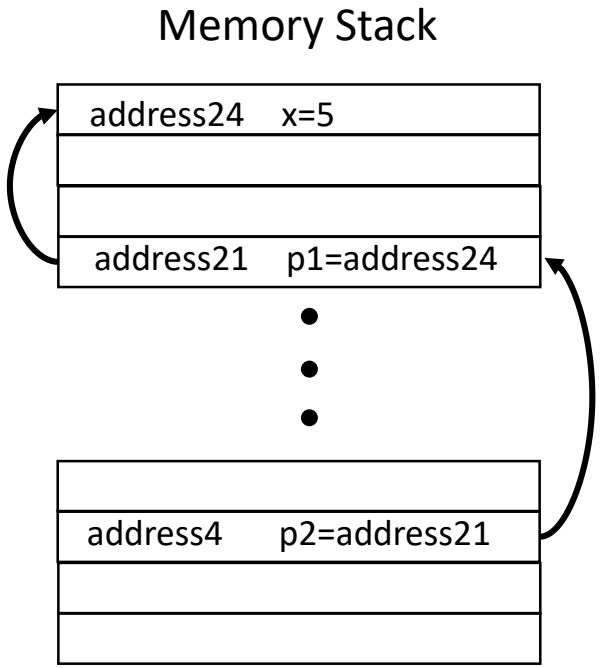


Pointers are also in memory! Their memory locations has an address!

What is the result of *(*p2)?

Pointer of a pointer?

```
int x = 5;
int *p1 = &x;
int **p2 = &p1;
```



Pointers are also in memory! Their memory locations has an address!

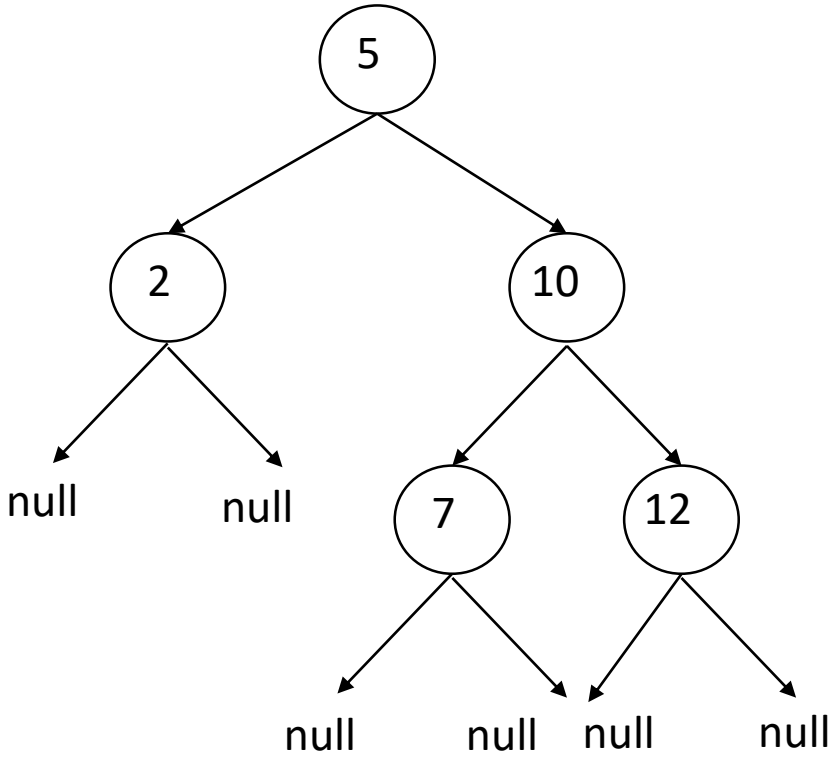
What is the result of *(*p2)?

Answer: *(*p2) = *(address1) = x = 5

The Null pointer

Does node 1 exists in the current tree?

What about initializations?



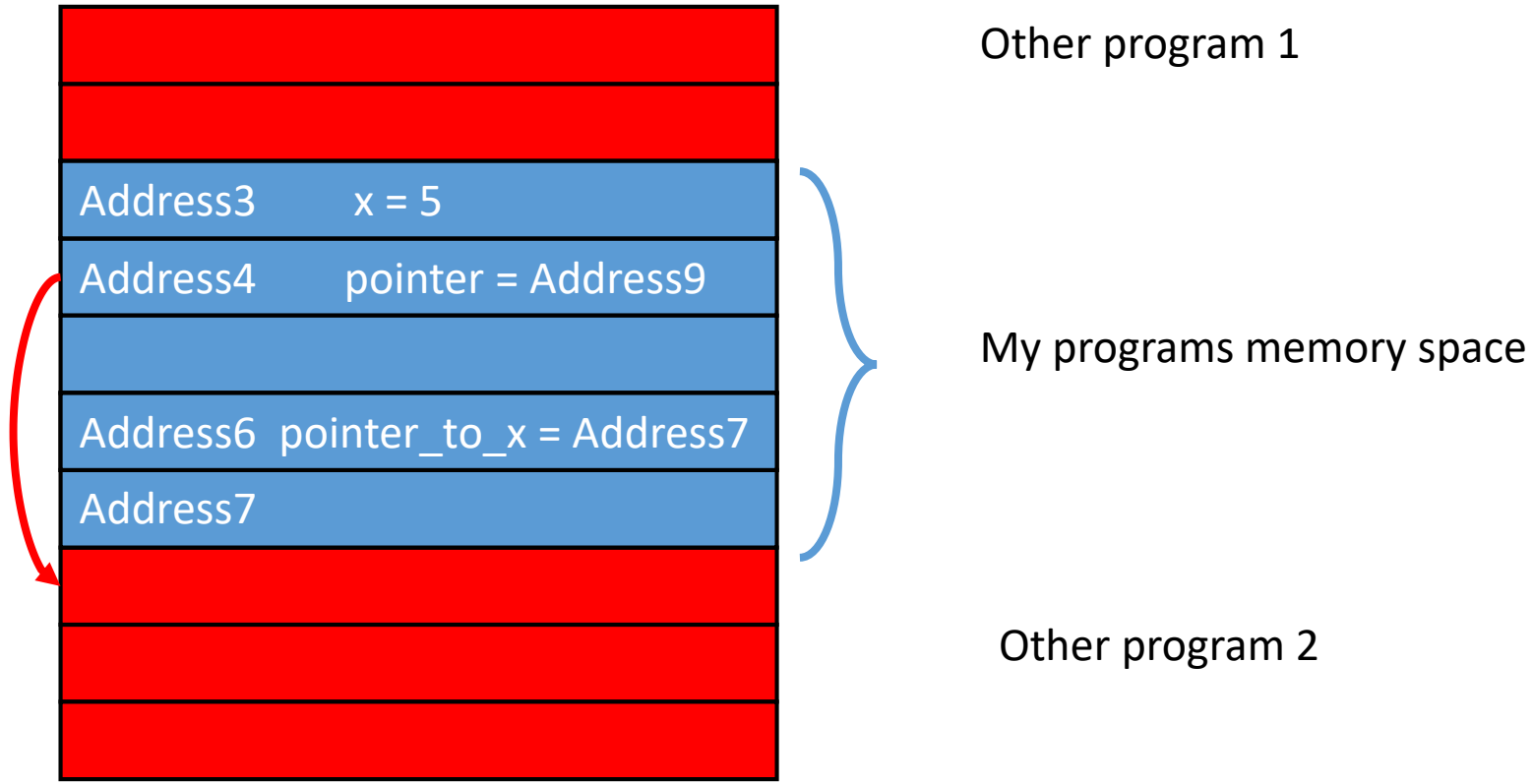
Segmentation Fault (SF)

This is a conceptional example

Trying to access memory locations that do not belong in my program lead to SF

Which of the references below would cause a SF?

1) int x2=*pointer;
2) int **x2 = &pointer;
3) int *x2 = pointer + 2;
4) int x2 = *(&x + 10);



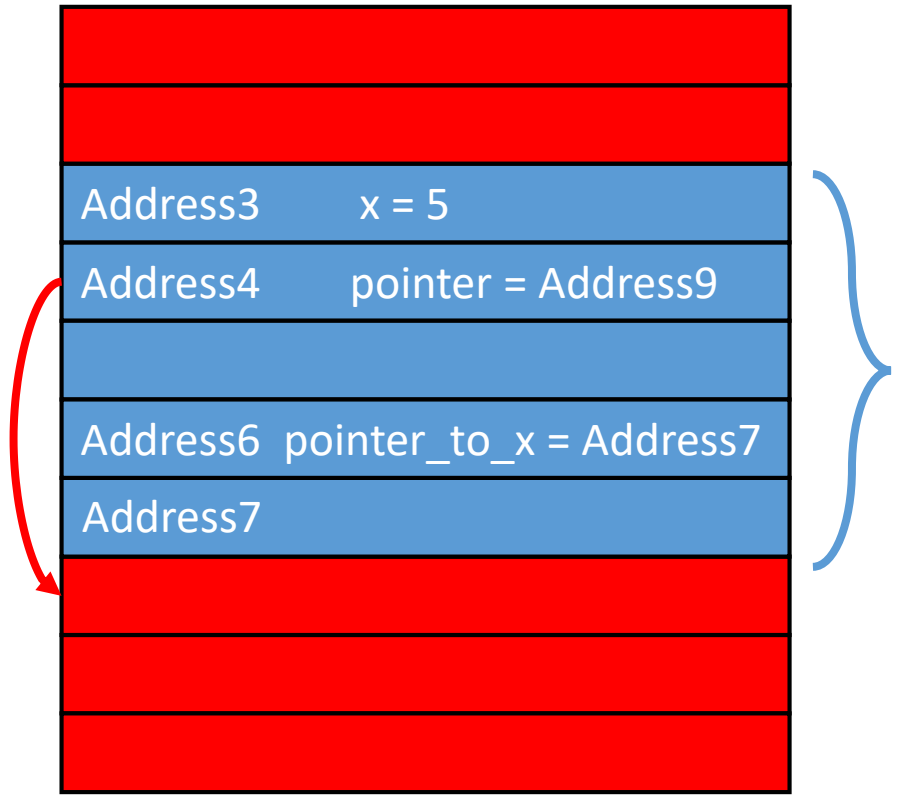
Segmentation Fault (SF)

This is a conceptional example

Trying to access memory locations that do not belong in my program lead to SF

Which of the references below would cause a SF?

1) int x2=*pointer; YES
2) int **x2 = &pointer; NO
3) int *x2 = pointer + 2; NO
4) int x2 = *(&x + 10); YES



Other program 1

My programs memory space

Other program 2

Exercise 1

- What is the result of the program below?

```
int  count = 10,
int *temp
int sum = 0;
temp = &count;
*temp = 20;
temp = &sum;
*temp = count;
printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum );
```

Exercise 1

- What is the result of the program below?

```
int  count = 10,
int *temp
int sum = 0;
temp = &count;
*temp = 20;
temp = &sum;
*temp = count;
printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum );
```

Answer: count = 20, *temp = 20, sum = 20

Exercise 2

```
void g(int x, int * y){
	printf("In g, x = %d, *y = %d\n",x,*y);
	x++;
	*y = *y - x;
	y = &x;
}

void f(int *a, int b){
	printf("In f, *a = %d, b = %d\n", *a, b);
	*a += b;
	b *= 2;
	g(*a,&b);
	printf("Back in f, *a = %d, b = %d\n",*a,b);
}

int main (void){
	int x = 3;
	int y = 4;
	f(&x, y);
	printf("In main: x = %d, y = %d\n", x, y);
	return (1);
}
```

Exercise 2

Answer:

In f, *a = 3, b = 4
In g, x = 7, *y = 8
Back in f, *a = 7, b = 0
In main: x = 7, y = 4