CS 50:
Software Design and Implementation

Pointers

Agenda

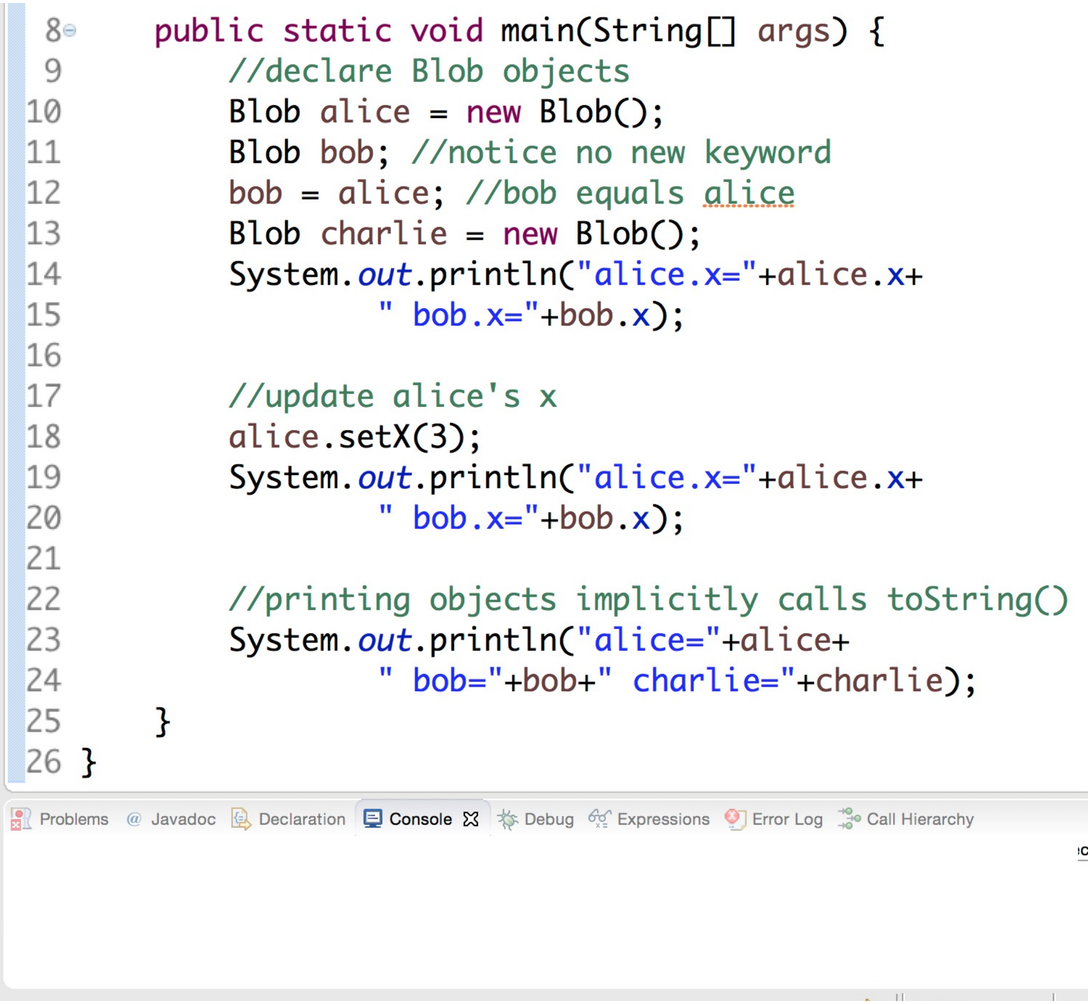
1. You’ve seen the *idea* of pointers in Java

2. C pointers

3. Pass by value

4. Activity

Declaring objects makes pointer on the stack, but object itself is elsewhere

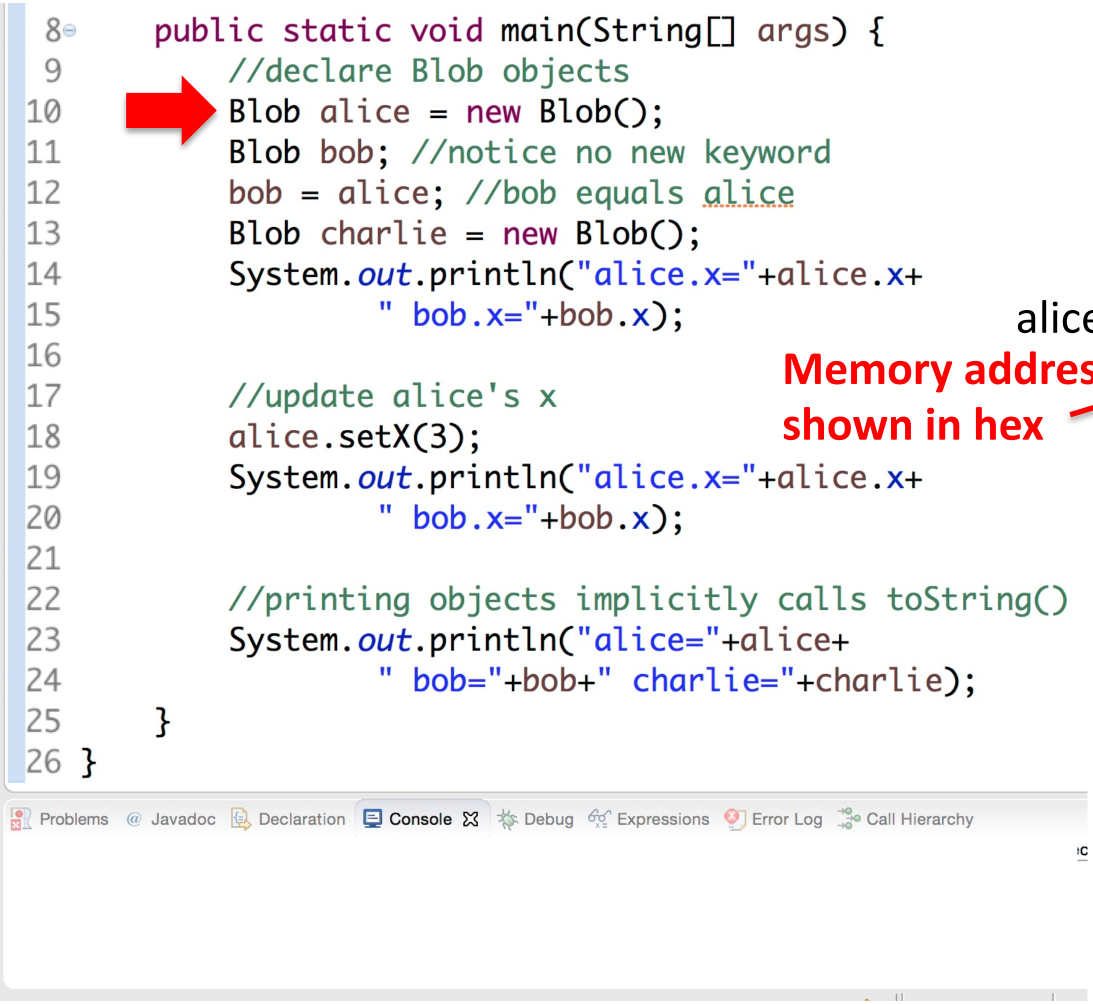


```
    public static void main(String[] args) {
        //declare Blob objects
        Blob alice = new Blob();
        Blob bob; //notice no new keyword
        bob = alice; //bob equals alice
        Blob charlie = new Blob();
        System.out.println("alice.x="+alice.x+
                " bob.x="+bob.x);

        //update alice's x
        alice.setX(3);
        System.out.println("alice.x="+alice.x+
                " bob.x="+bob.x);

        //printing objects implicitly calls toString()
        System.out.println("alice="+alice+
                " bob="+bob+" charlie="+charlie);
    }
}
```

Stack

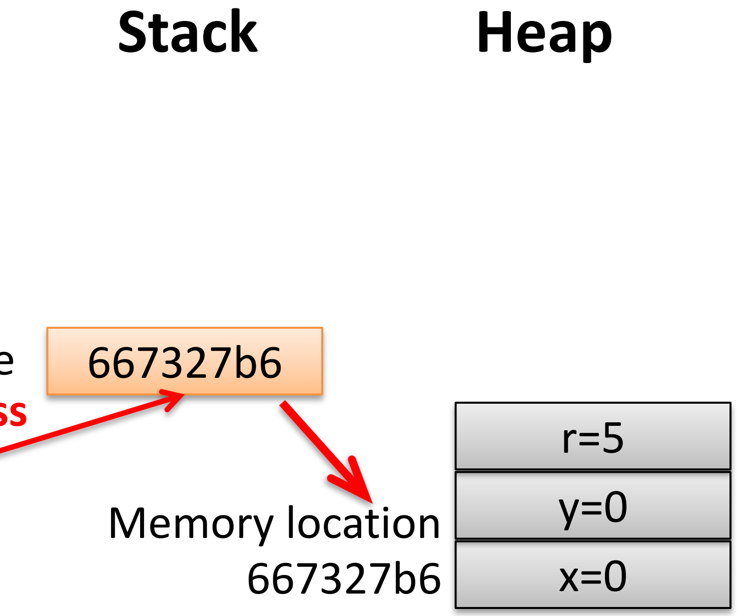
Heap

Declaring objects makes pointer on the stack, but object itself is elsewhere



```
 8     public static void main(String[] args) {
 9         //declare Blob objects
10         Blob alice = new Blob();
11         Blob bob; //notice no new keyword
12         bob = alice; //bob equals alice
13         Blob charlie = new Blob();
14         System.out.println("alice.x="+alice.x+
15                 " bob.x="+bob.x);
16 
17         //update alice's x
18         alice.setX(3);
19         System.out.println("alice.x="+alice.x+
20                 " bob.x="+bob.x);
21 
22         //printing objects implicitly calls toString()
23         System.out.println("alice="+alice+
24                 " bob="+bob+" charlie="+charlie);
25     }
26 }
```

Memory address shown in hex



Stack

Heap

alice

667327b6

Memory location 667327b6

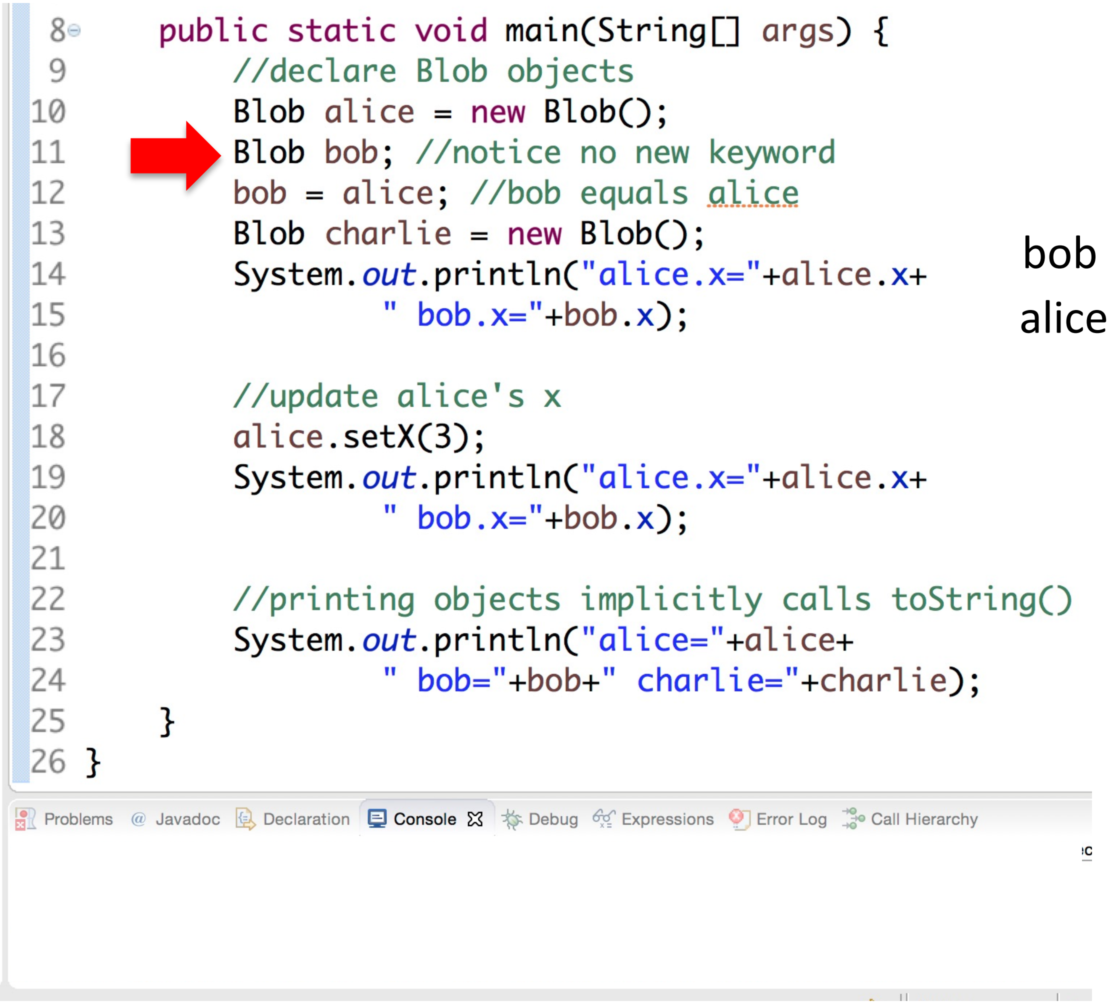
r=5

y=0

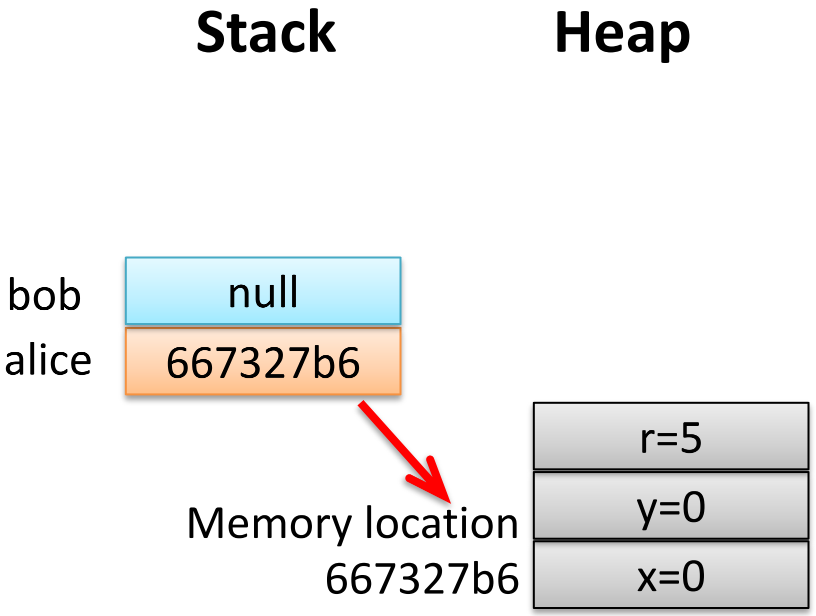
x=0

- After line 10, stack holds ***memory address*** of object (with primitives, stack holds variable's *value*)
- Memory address tells Java where to find the "alice" object in memory
- Object itself allocated elsewhere in memory (in heap, not on stack)
- OS chooses where to allocate

Declaring objects makes pointer on the stack, but object itself is elsewhere



```
 8      public static void main(String[] args) {
 9          //declare Blob objects
10          Blob alice = new Blob();
11          Blob bob; //notice no new keyword
12          bob = alice; //bob equals alice
13          Blob charlie = new Blob();
14          System.out.println("alice.x="+alice.x+
15                  " bob.x="+bob.x);
16
17          //update alice's x
18          alice.setX(3);
19          System.out.println("alice.x="+alice.x+
20                  " bob.x="+bob.x);
21
22          //printing objects implicitly calls toString()
23          System.out.println("alice="+alice+
24                  " bob="+bob+" charlie="+charlie);
25      }
26 }
```



Stack

Heap

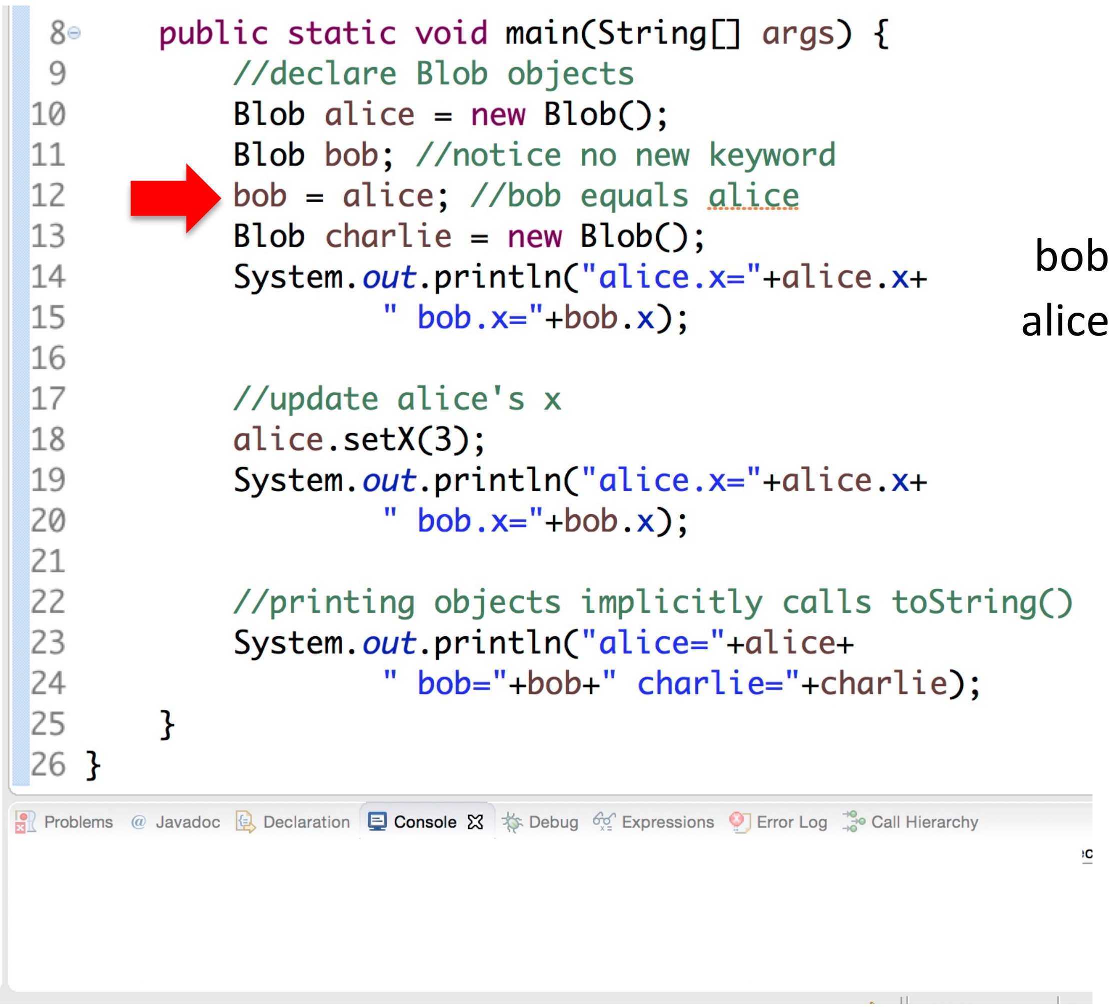
bob	null
alice	667327b6

Memory location 667327b6

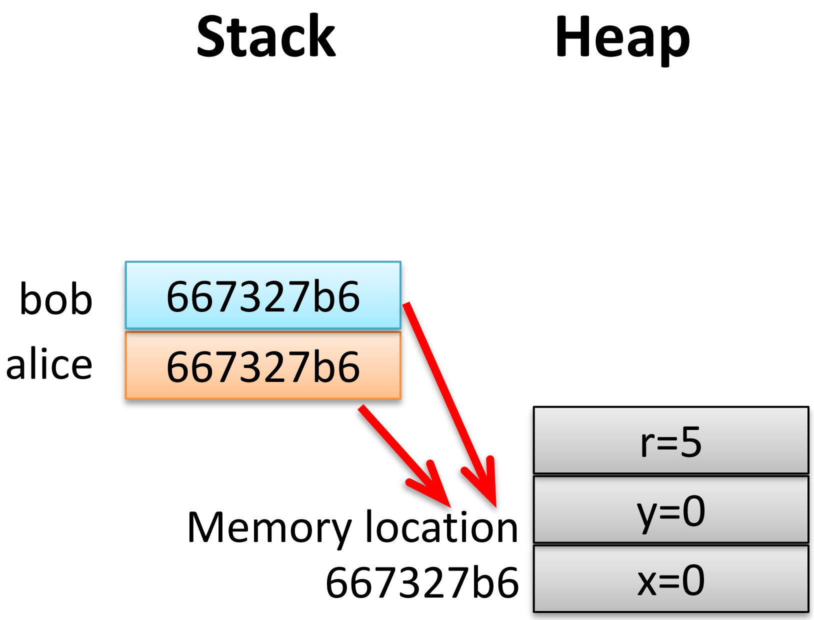
r=5
y=0
x=0

- **After line 11, “bob” is allocated on the stack, but is null (points nowhere)**
- **This is because bob did not use the “new” keyword**
- **Null pointer exception if try to use bob now**

Declaring objects makes pointer on the stack, but object itself is elsewhere

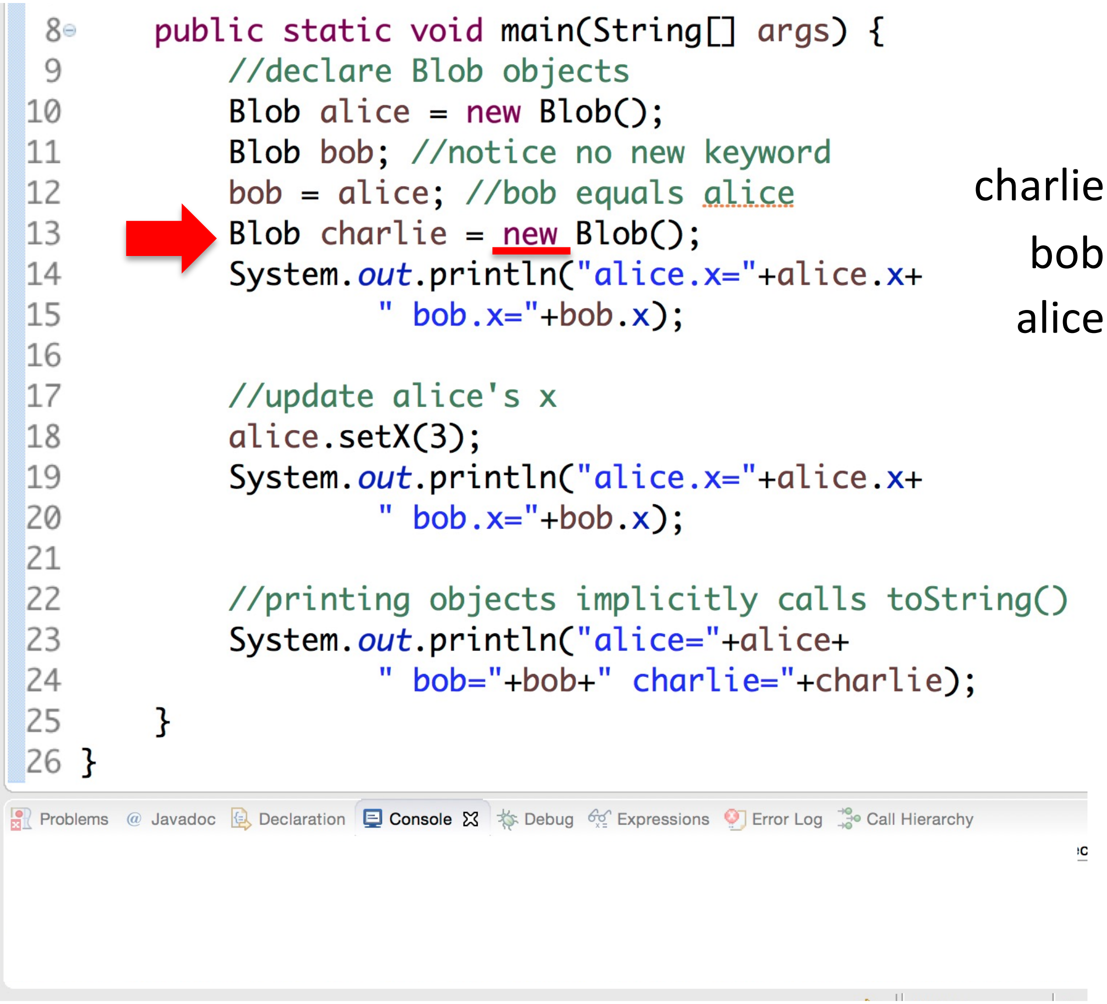


```
 8      public static void main(String[] args) {
 9          //declare Blob objects
10          Blob alice = new Blob();
11          Blob bob; //notice no new keyword
12          bob = alice; //bob equals alice
13          Blob charlie = new Blob();
14          System.out.println("alice.x="+alice.x+
15                  " bob.x="+bob.x);
16
17          //update alice's x
18          alice.setX(3);
19          System.out.println("alice.x="+alice.x+
20                  " bob.x="+bob.x);
21
22          //printing objects implicitly calls toString()
23          System.out.println("alice="+alice+
24                  " bob="+bob+" charlie="+charlie);
25      }
26  }
```

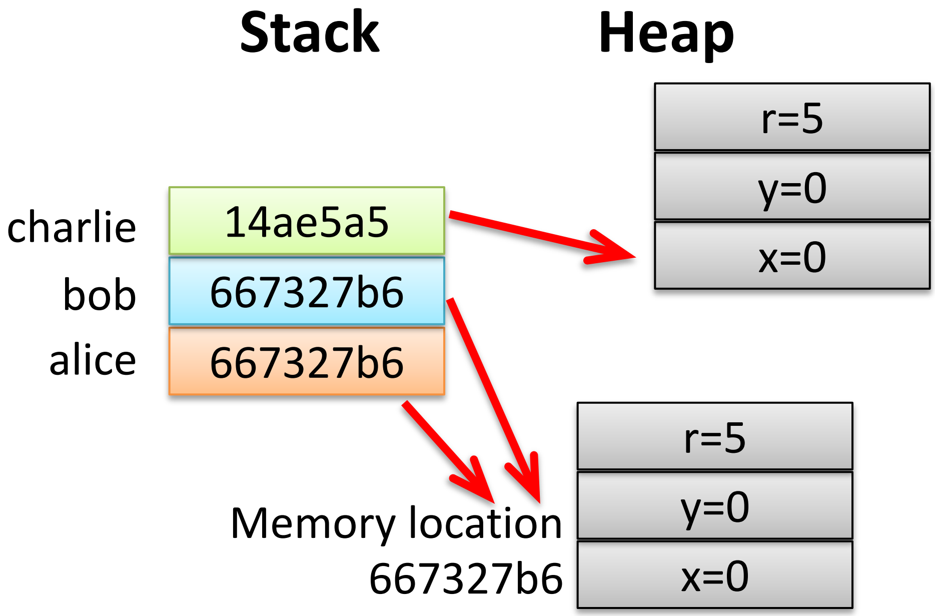


- **Line 12, bob set equal to alice**
- **bob gets same value on stack that alice holds**
- **bob now points to the exact same memory location as alice**
- **bob and alice are “aliases” of each other**

Declaring objects makes pointer on the stack, but object itself is elsewhere



```
public static void main(String[] args) {
    //declare Blob objects
    Blob alice = new Blob();
    Blob bob; //notice no new keyword
    bob = alice; //bob equals alice
    Blob charlie = new Blob();
    System.out.println("alice.x="+alice.x+
            " bob.x="+bob.x);

    //update alice's x
    alice.setX(3);
    System.out.println("alice.x="+alice.x+
            " bob.x="+bob.x);

    //printing objects implicitly calls toString()
    System.out.println("alice="+alice+
            " bob="+bob+" charlie="+charlie);
}
}
```



Stack

Heap

charlie	14ae5a5
bob	667327b6
alice	667327b6

r=5

y=0

x=0

Memory location 667327b6

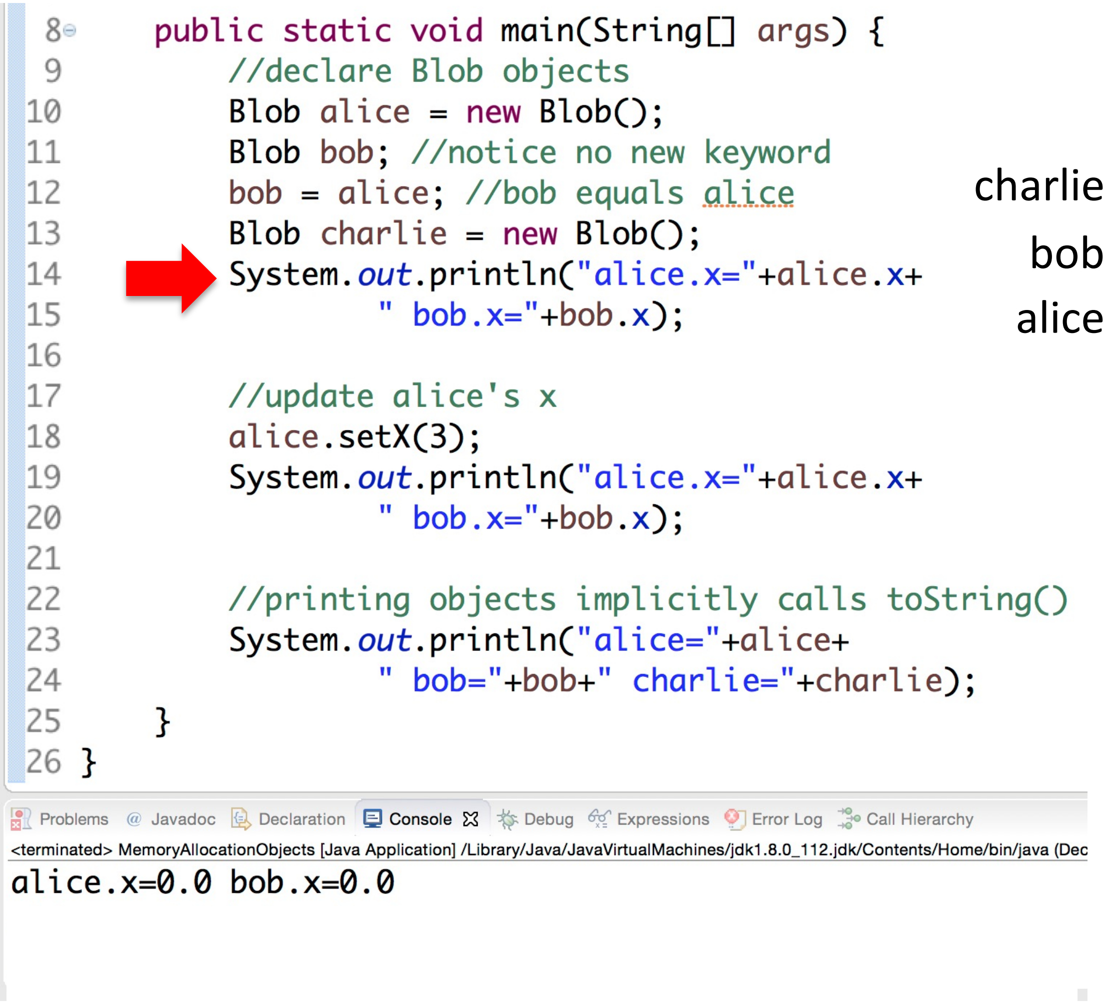
r=5

y=0

x=0

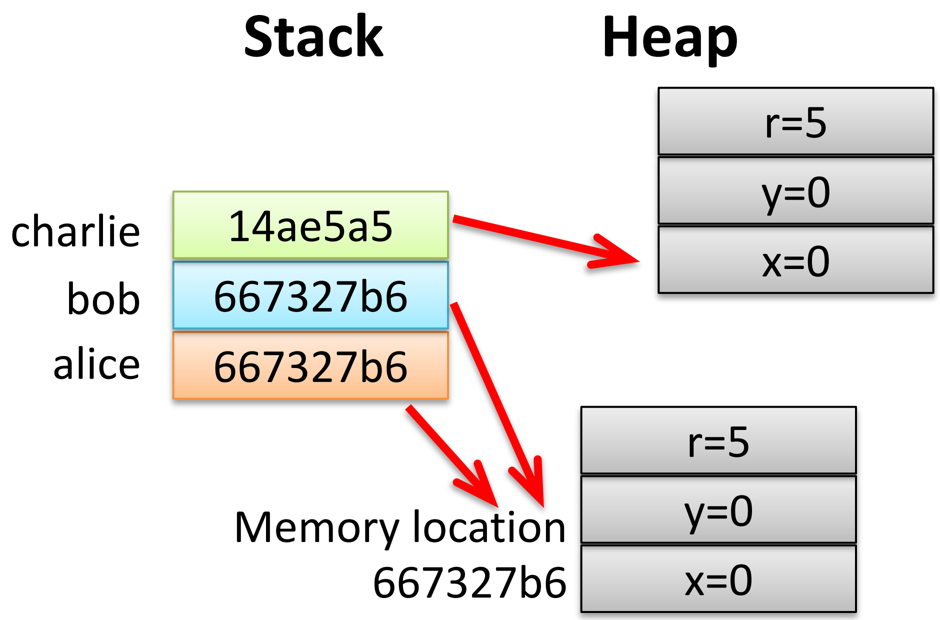
Charlie object gets new allocation elsewhere in memory because "new" keyword used

Declaring objects makes pointer on the stack, but object itself is elsewhere



```
    public static void main(String[] args) {
        //declare Blob objects
        Blob alice = new Blob();
        Blob bob; //notice no new keyword
        bob = alice; //bob equals alice
        Blob charlie = new Blob();
        System.out.println("alice.x="+alice.x+
                " bob.x="+bob.x);

        //update alice's x
        alice.setX(3);
        System.out.println("alice.x="+alice.x+
                " bob.x="+bob.x);

        //printing objects implicitly calls toString()
        System.out.println("alice="+alice+
                " bob="+bob+" charlie="+charlie);
    }
}
```

```
<terminated> MemoryAllocationObjects [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec
alice.x=0.0 bob.x=0.0
```



Stack

Heap

charlie 14ae5a5

bob 667327b6

alice 667327b6

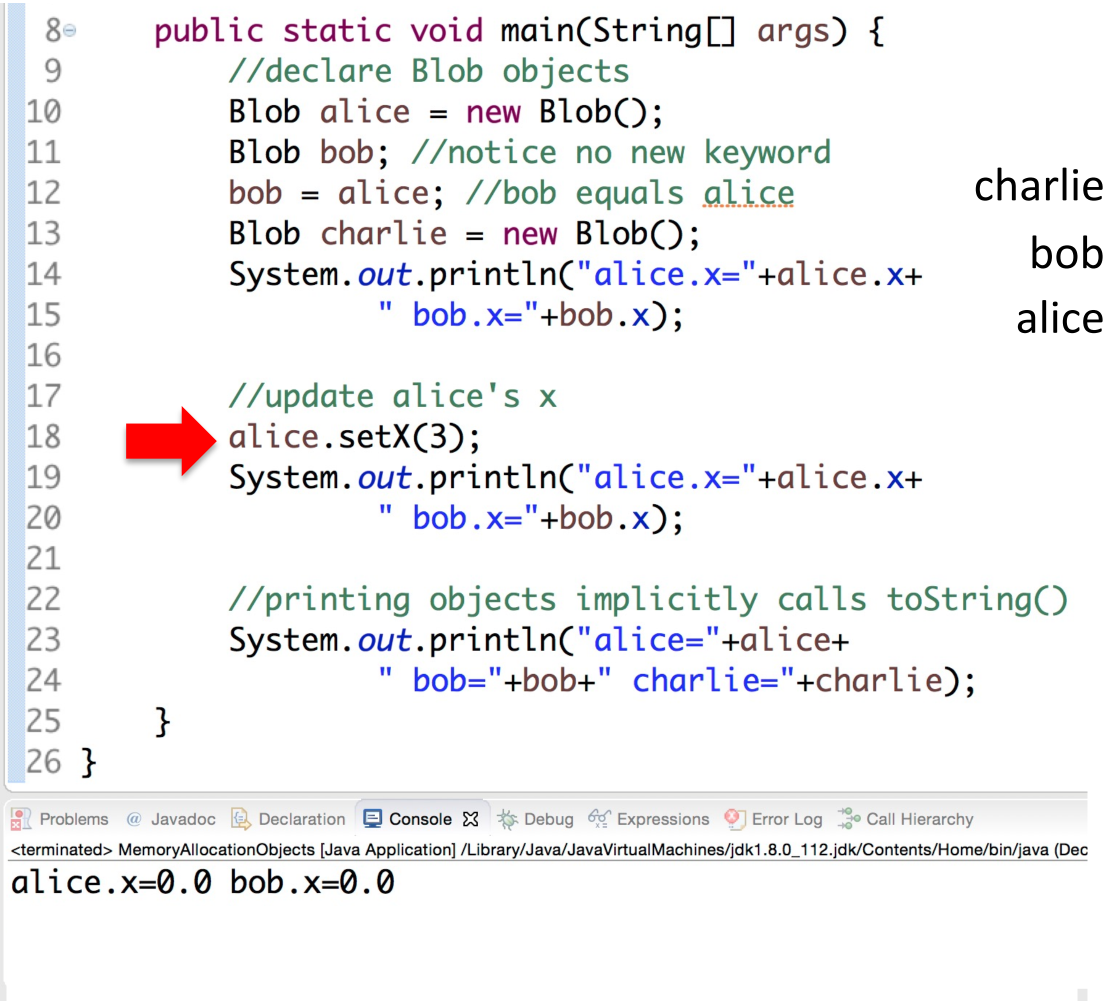
r=5
y=0
x=0

r=5
y=0
x=0

Memory location
667327b6

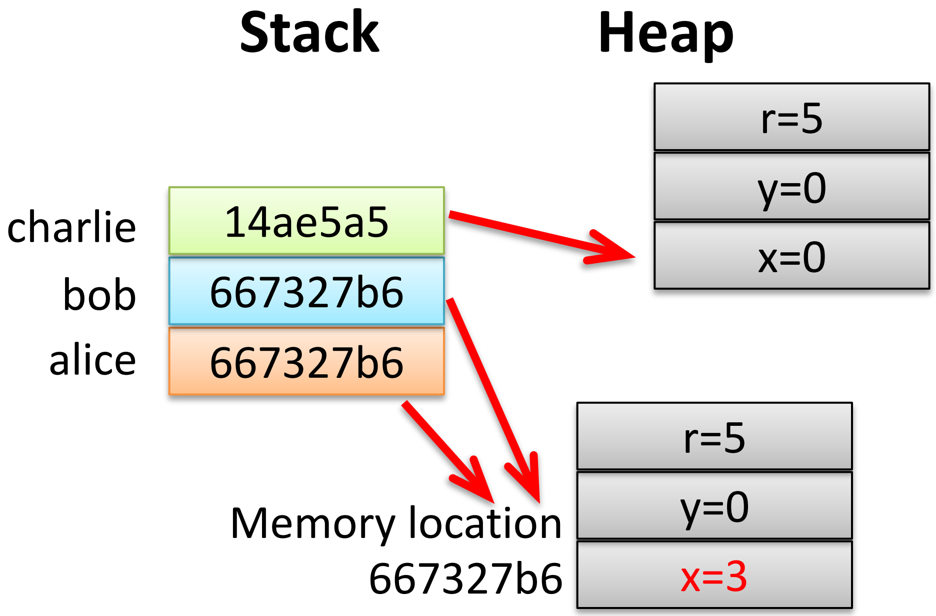
x value for alice and bob is the same because stored at the exact same memory address

Declaring objects makes pointer on the stack, but object itself is elsewhere



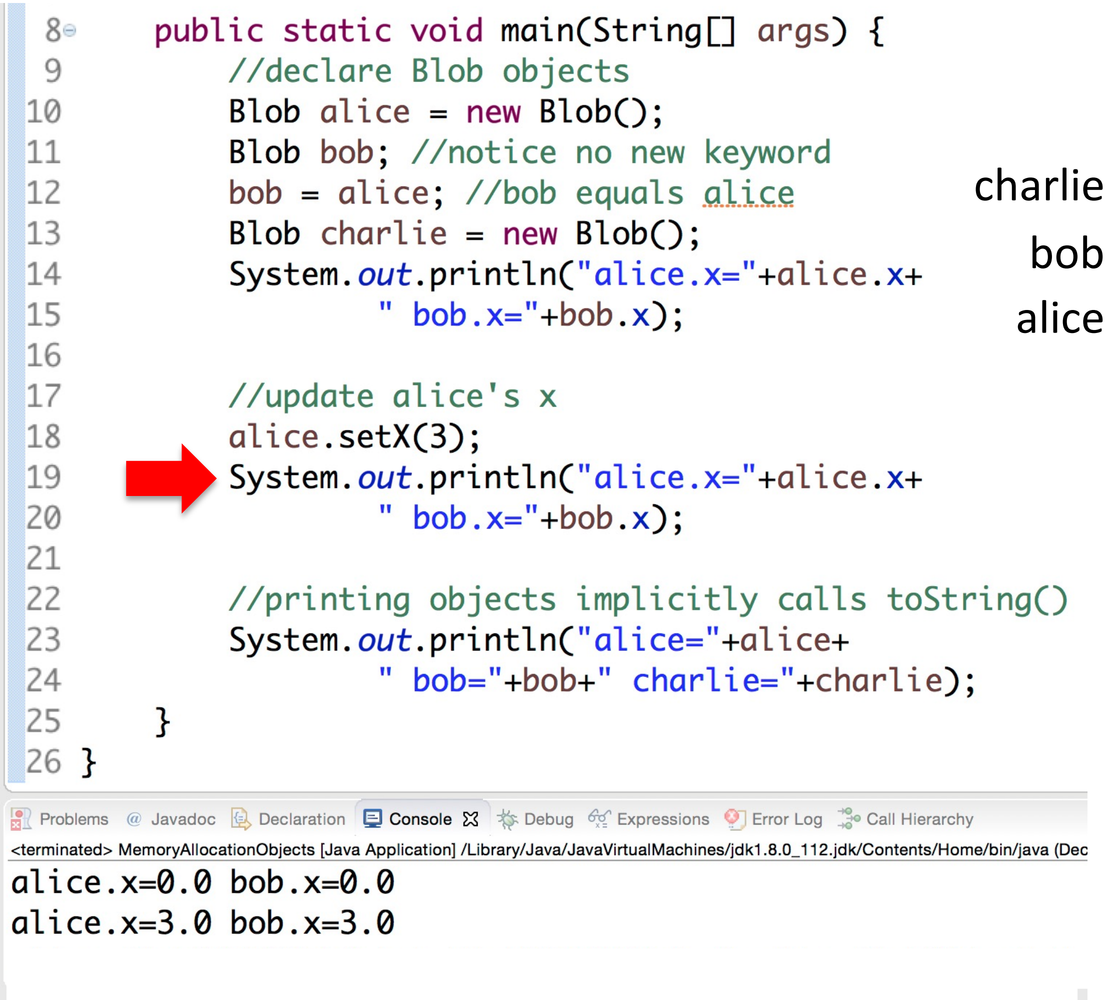
```
    public static void main(String[] args) {
        //declare Blob objects
        Blob alice = new Blob();
        Blob bob; //notice no new keyword
        bob = alice; //bob equals alice
        Blob charlie = new Blob();
        System.out.println("alice.x="+alice.x+
                " bob.x="+bob.x);

        //update alice's x
        alice.setX(3);
        System.out.println("alice.x="+alice.x+
                " bob.x="+bob.x);

        //printing objects implicitly calls toString()
        System.out.println("alice="+alice+
                " bob="+bob+" charlie="+charlie);
    }
}
```

```
<terminated> MemoryAllocationObjects [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec
alice.x=0.0 bob.x=0.0
```



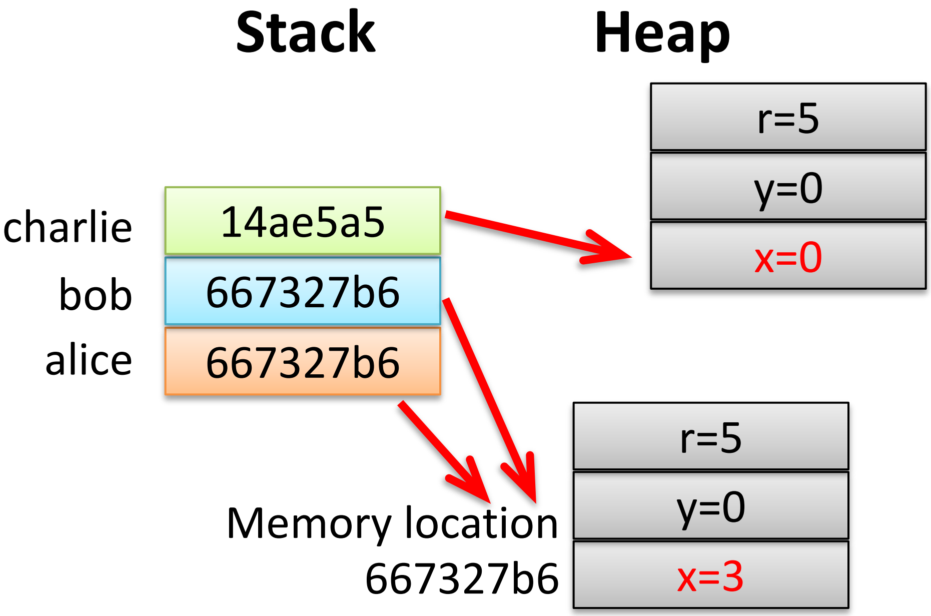
- **alice.x set to 3**
- **What is bob.x?**

Declaring objects makes pointer on the stack, but object itself is elsewhere



```java
    public static void main(String[] args) {
        //declare Blob objects
        Blob alice = new Blob();
        Blob bob; //notice no new keyword
        bob = alice; //bob equals alice
        Blob charlie = new Blob();
        System.out.println("alice.x="+alice.x+
                " bob.x="+bob.x);

        //update alice's x
        alice.setX(3);
        System.out.println("alice.x="+alice.x+
                " bob.x="+bob.x);

        //printing objects implicitly calls toString()
        System.out.println("alice="+alice+
                " bob="+bob+" charlie="+charlie);
    }
}
```

```
<terminated> MemoryAllocationObjects [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec
alice.x=0.0 bob.x=0.0
alice.x=3.0 bob.x=3.0
```



- **x is the same for both alice and bob objects because they point to the same memory address (called alias)**
- **Like Python setting two lists equal to each other, change one list, change the other also**
- **Charlie's x is still 0**

Agenda

1. You’ve seen the *idea* of pointers in Java
2. C pointers
3. Pass by value
4. Activity

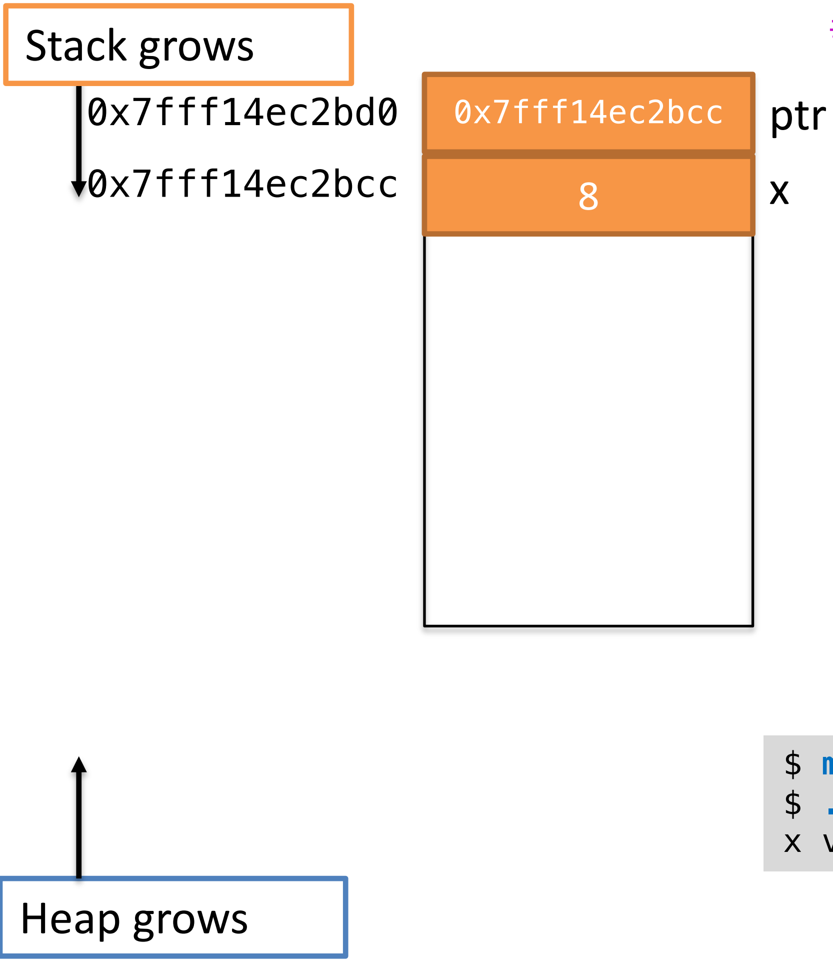
Pointers can also act as an alias. We saw something similar in CS 10

alias.c

ptr holds address of x

```
#include<stdio.h>

int main() {
        int x = 8;
        int *ptr = &x;   //ptr get address of x

        printf("x value: %d addr: %p\n",x,(void *)&x);
        printf("ptr points to %p addr: %p value: %d\n",
                (void *)ptr, (void *)&ptr, *ptr);

        //increment ptr by 1
        *ptr = 9;

        printf("x value %d\n",x);
        printf("ptr value %d\n",*ptr);

        return 0;
}
```

Pointers can also act as an alias. We saw something similar in CS 10



Stack grows

0x7fff14ec2bd0

0x7fff14ec2bcc

0x7fff14ec2bcc

ptr

8

x

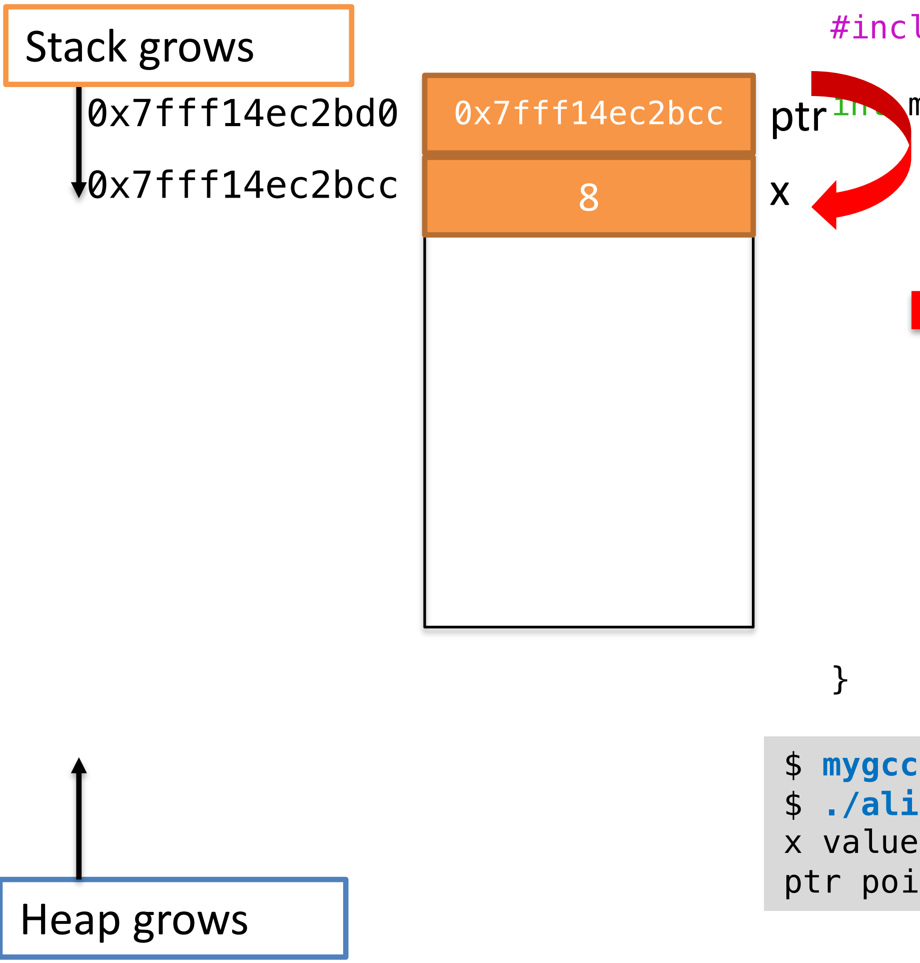
Heap grows

alias.c

```
#include<stdio.h>

int main() {
        int x = 8;
        int *ptr = &x;    //ptr get address of x

        printf("x value: %d addr: %p\n",x,(void *)&x);
        printf("ptr points to %p addr: %p value: %d\n",
                (void *)ptr, (void *)&ptr, *ptr);

        //increment ptr by 1
        *ptr = 9;

        printf("x value %d\n",x);
        printf("ptr value %d\n",*ptr);

        return 0;
}
```

```
$ mygcc -o alias alias.c
$ ./alias
x value: 8 addr: 0x7fff14ec2bcc
```

Pointers can also act as an alias. We saw something similar in CS 10

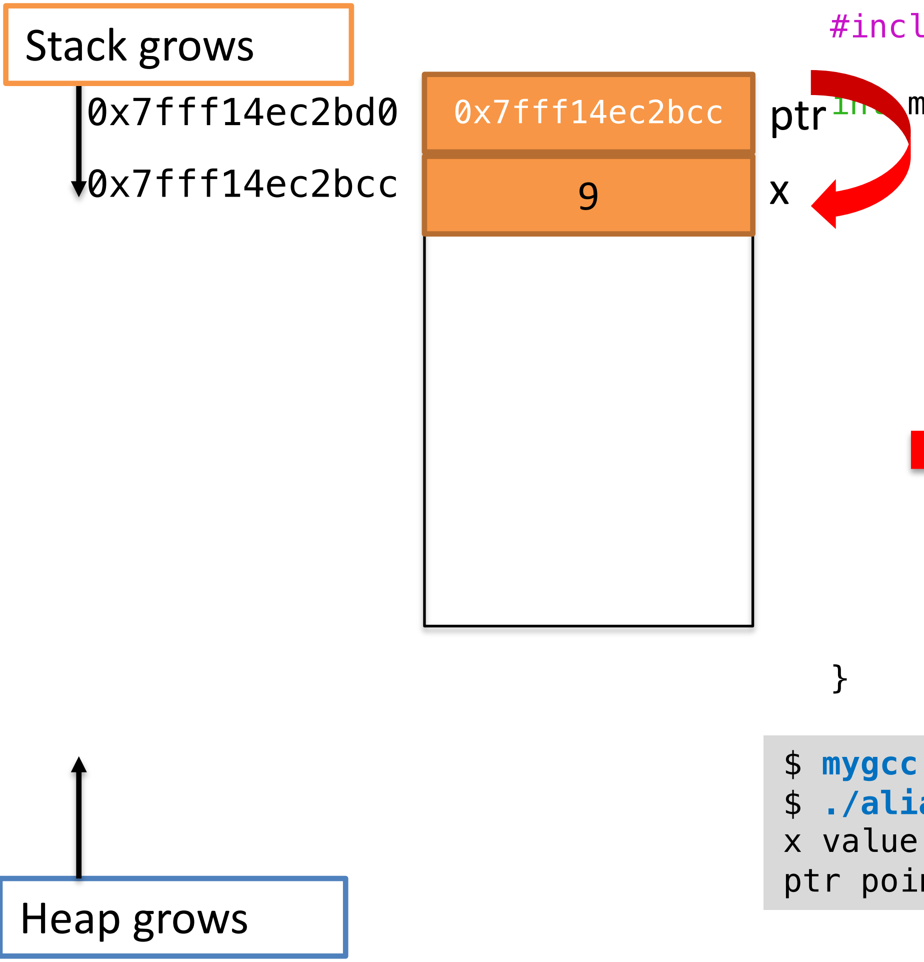


alias.c

```
#include<stdio.h>

int main() {
        int x = 8;
        int *ptr = &x;    //ptr get address of x

        printf("x value: %d addr: %p\n",x,(void *)&x);
        printf("ptr points to %p addr: %p value: %d\n",
                (void *)ptr, (void *)&ptr, *ptr);

        //increment ptr by 1
        *ptr = 9;

        printf("x value %d\n",x);
        printf("ptr value %d\n",*ptr);

        return 0;
}
```

```
$ mygcc -o alias alias.c
$ ./alias
x value: 8 addr: 0x7fff14ec2bcc
ptr points to 0x7fff14ec2bcc addr: 0x7fff14ec2bd0 value: 8
```

Pointers can also act as an alias. We saw something similar in CS 10



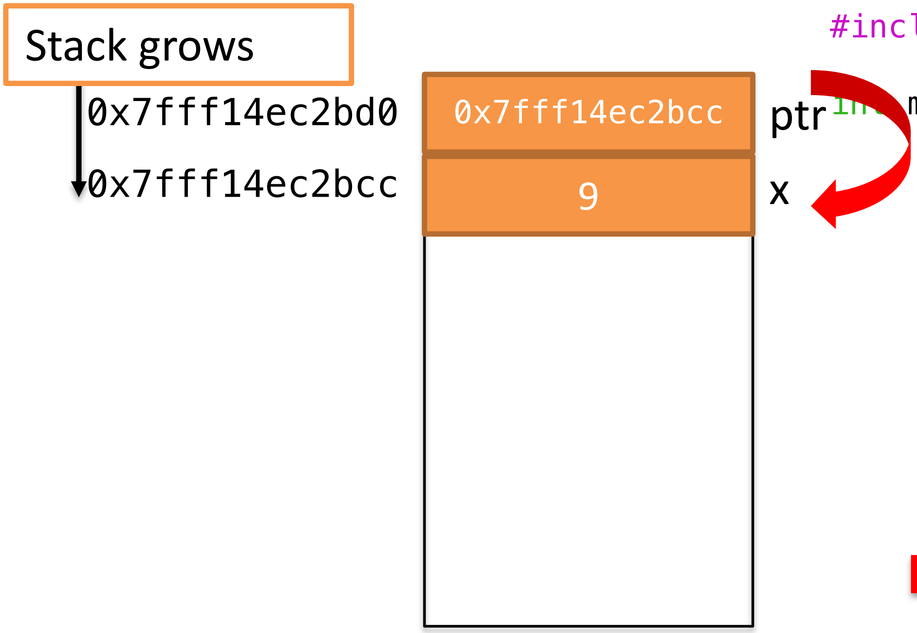
alias.c

```
#include<stdio.h>

int main() {
        int x = 8;
        int *ptr = &x;    //ptr get address of x

        printf("x value: %d addr: %p\n",x,(void *)&x);
        printf("ptr points to %p addr: %p value: %d\n",
                (void *)ptr, (void *)&ptr, *ptr);

        //increment ptr by 1
        *ptr = 9;

        printf("x value %d\n",x);
        printf("ptr value %d\n",*ptr);

        return 0;
}
```

Set value of deferenced memory location to 9 by using *p

```
$ mygcc -o alias alias.c
$ ./alias
x value: 8 addr: 0x7fff14ec2bcc
ptr points to 0x7fff14ec2bcc addr: 0x7fff14ec2bd0 value: 8
```

Pointers can also act as an alias. We saw something similar in CS 10



alias.c

```
#include<stdio.h>

int main() {
        int x = 8;
        int *ptr = &x;    //ptr get address of x

        printf("x value: %d addr: %p\n",x,(void *)&x);
        printf("ptr points to %p addr: %p value: %d\n",
                (void *)ptr, (void *)&ptr, *ptr);

        //increment ptr by 1
        *ptr = 9;

        printf("x value %d\n",x);
        printf("ptr value %d\n",*ptr);

        return 0;
}
```

Set value of deferenced memory location to 9 by using *p

Heap grows

Because ptr points to x, changing the value of ptr (using *ptr) changes x also!

```
$ mygcc -o alias alias.c
$ ./alias
x value: 8 addr: 0x7fff14ec2bcc
ptr points to 0x7fff14ec2bcc addr: 0x7fff14ec2bd0 value: 8
x value 9
ptr value 9
```

Agenda

1. You’ve seen the *idea* of pointers in Java

2. C pointers

➡ 3. Pass by value

4. Activity

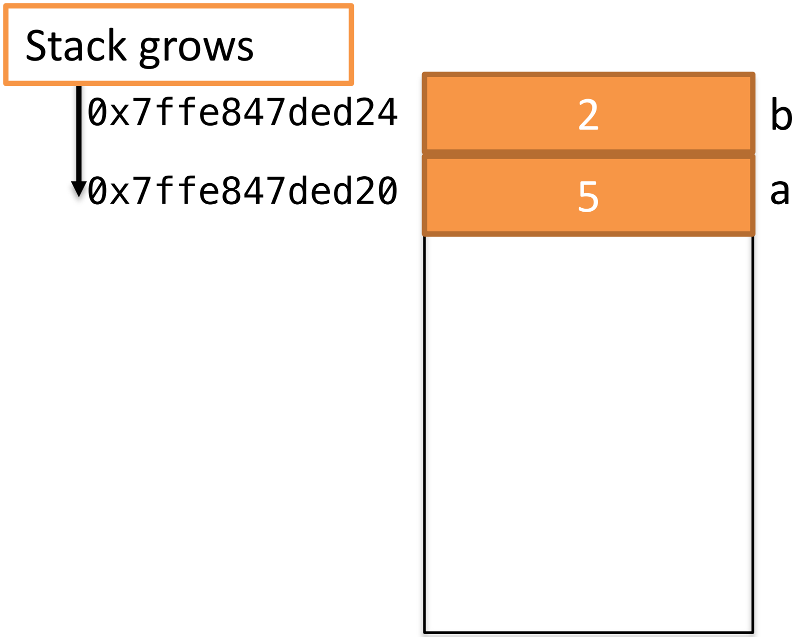
C passes values to by using “pass by value”, creates a copy of parameters on the stack

Goal call swap and exchange values in a and b

swap0.c

```c
void swap (int a, int b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            a, b, temp);
    temp = a;
    a = b;
    b = temp;

    printf("\tswapped values: a=%d and b=%d\n", a, b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n", a, b);

    printf("Calling swap function\n");
    swap(a, b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n", a, b);

    return 0;
}
```

Local variables a and b in main pushed onto stack

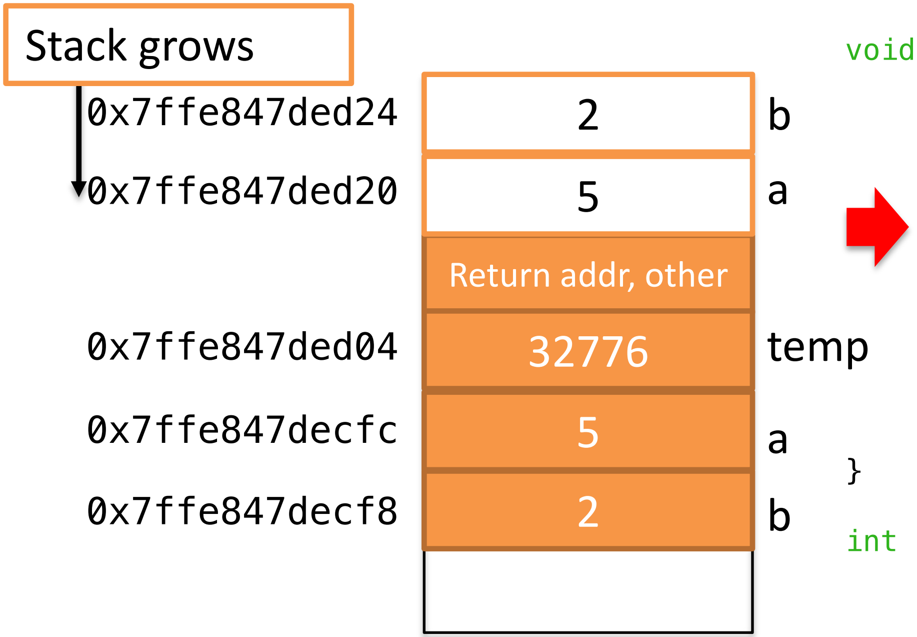


swap0.c

```
void swap (int a, int b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            a, b, temp);
    temp = a;
    a = b;
    b = temp;

    printf("\tswapped values: a=%d and b=%d\n", a, b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n", a, b);

    printf("Calling swap function\n");
    swap(a, b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n", a, b);

    return 0;
}
```

```
$ mygcc –o swap swap.c
$ ./swap
In main
addresses a=0x7ffe847ded20, b=0x7ffe847ded24
values a=5 and b=2
Calling swap function
```

Calling swap passes copy of a and b to func, values pushed onto stack



swap0.c

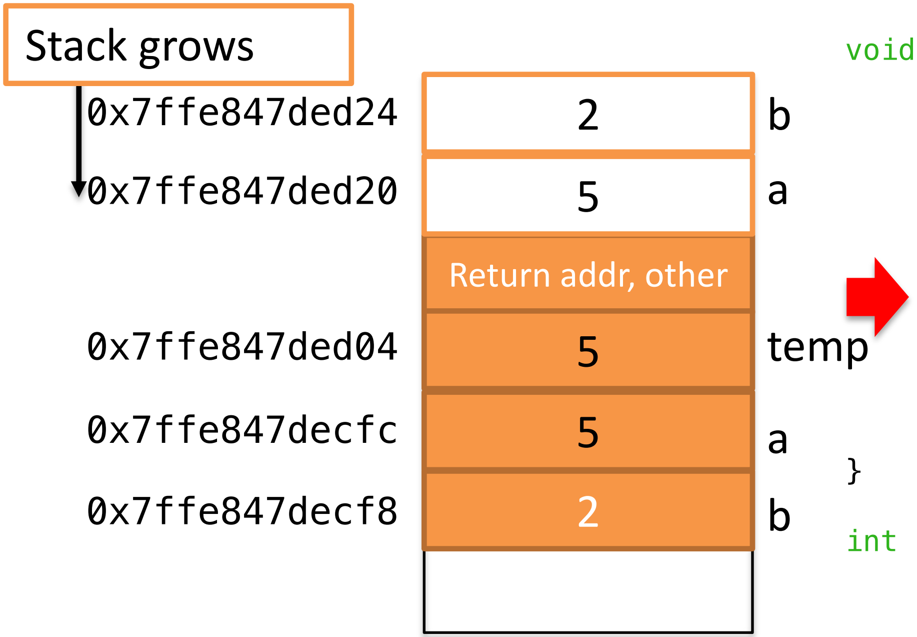
```c
void swap (int a, int b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            a,b,temp);
    temp = a;
    a = b;
    b = temp;

    printf("\tswapped values: a=%d and b=%d\n",a,b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(a, b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

Local variables are not initialized

Temp holds random value

```
$ mygcc -o swap swap.c
$ ./swap
In main
addresses a=0x7ffe847ded20, b=0x7ffe847ded24
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffe847decfc, b=0x7ffe847decf8,
temp=0x7ffe847ded04
parameters values: a=5, b=2, temp=32766
```

Swap works as expected in func, exchanging local copy of a and b



Stack grows

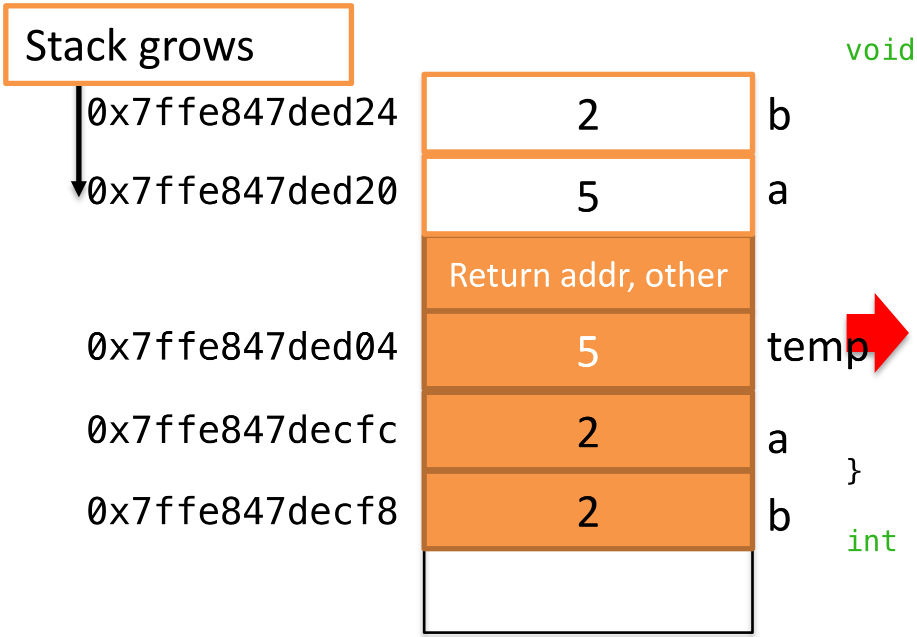
0x7ffe847ded24	2	b
0x7ffe847ded20	5	a
	Return addr, other	
0x7ffe847ded04	5	temp
0x7ffe847decfc	5	a
0x7ffe847decf8	2	b

swap0.c

```c
void swap (int a, int b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            a,b,temp);
    temp = a;
    a = b;
    b = temp;

    printf("\tswapped values: a=%d and b=%d\n",a,b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(a, b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

```
$ mygcc –o swap swap.c
$ ./swap
In main
addresses a=0x7ffe847ded20, b=0x7ffe847ded24
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffe847decfc, b=0x7ffe847decf8,
temp=0x7ffe847ded04
parameters values: a=5, b=2, temp=32766
```

Swap works as expected in func, exchanging local copy of a and b



Stack grows

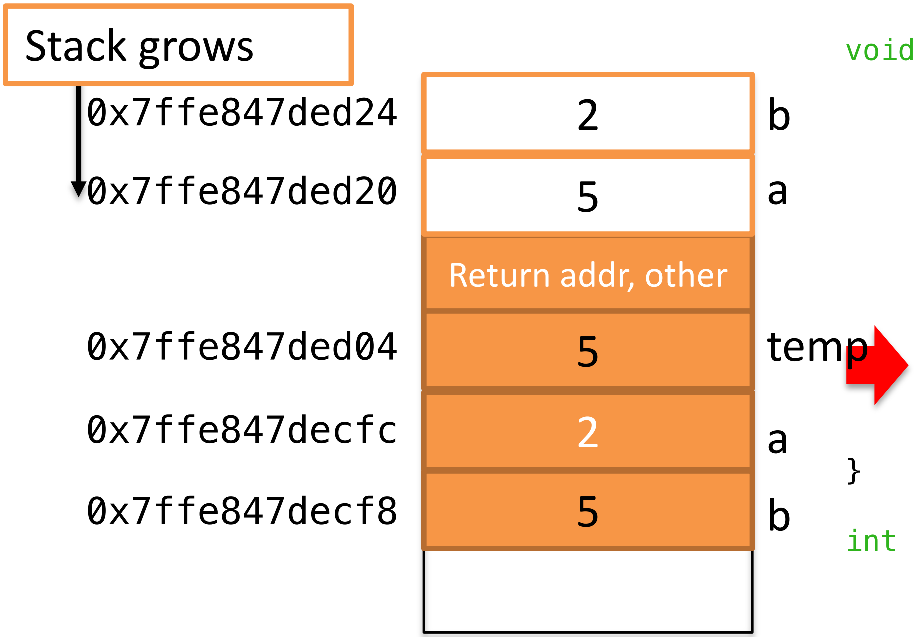
0x7ffe847ded24	2	b
0x7ffe847ded20	5	a
	Return addr, other	
0x7ffe847ded04	5	temp
0x7ffe847decfc	2	a
0x7ffe847decf8	2	b

swap0.c

```c
void swap (int a, int b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            a,b,temp);
    temp = a;
    a = b;
    b = temp;

    printf("\tswapped values: a=%d and b=%d\n",a,b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(a, b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

```
$ mygcc -o swap swap.c
$ ./swap
In main
addresses a=0x7ffe847ded20, b=0x7ffe847ded24
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffe847decfc, b=0x7ffe847decf8,
temp=0x7ffe847ded04
parameters values: a=5, b=2, temp=32766
```

Swap works as expected in func, exchanging local copy of a and b



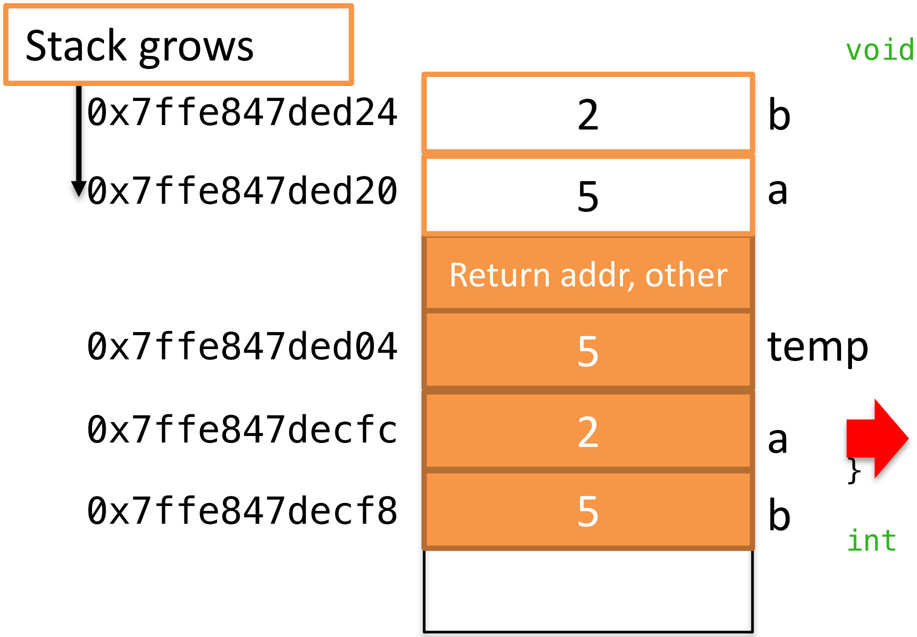
swap0.c

```
void swap (int a, int b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            a,b,temp);
    temp = a;
    a = b;
    b = temp;

    printf("\tswapped values: a=%d and b=%d\n",a,b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(a, b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

```
$ mygcc -o swap swap.c
$ ./swap
In main
addresses a=0x7ffe847ded20, b=0x7ffe847ded24
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffe847decfc, b=0x7ffe847decf8,
temp=0x7ffe847ded04
parameters values: a=5, b=2, temp=32766
```

Swap works as expected in func, exchanging local copy of a and b

swap0.c

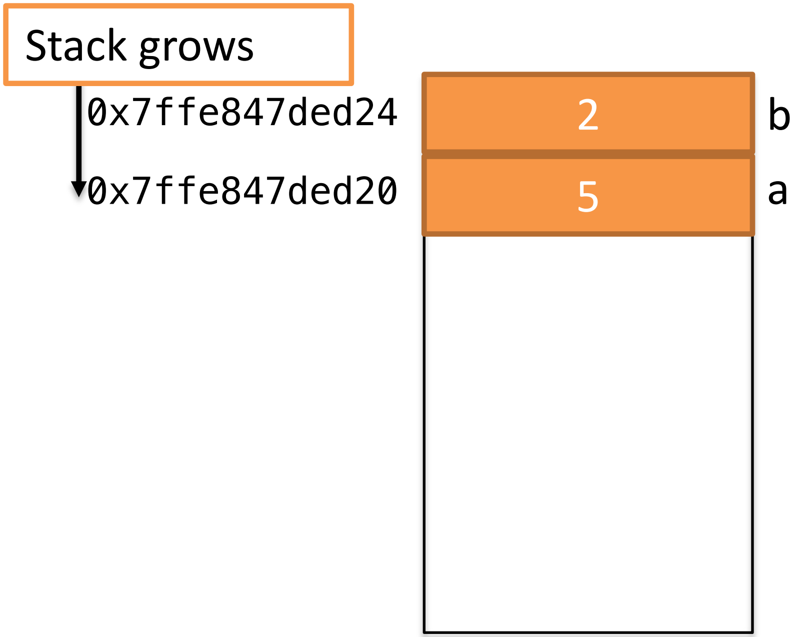


```
void swap (int a, int b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            a,b,temp);
    temp = a;
    a = b;
    b = temp;

    printf("\tswapped values: a=%d and b=%d\n",a,b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(a, b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

```
$ mygcc -o swap swap.c
$ ./swap
In main
addresses a=0x7ffe847ded20, b=0x7ffe847ded24
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffe847decfc, b=0x7ffe847decf8,
temp=0x7ffe847ded04
parameters values: a=5, b=2, temp=32766
swapped values: a=2 and b=5
```

Swap worked as expected!

When function ends, pop stack to remove local variables, return address, parameters



Stack grows

Address	Value	Variable
0x7ffe847ded24	2	b
0x7ffe847ded20	5	a

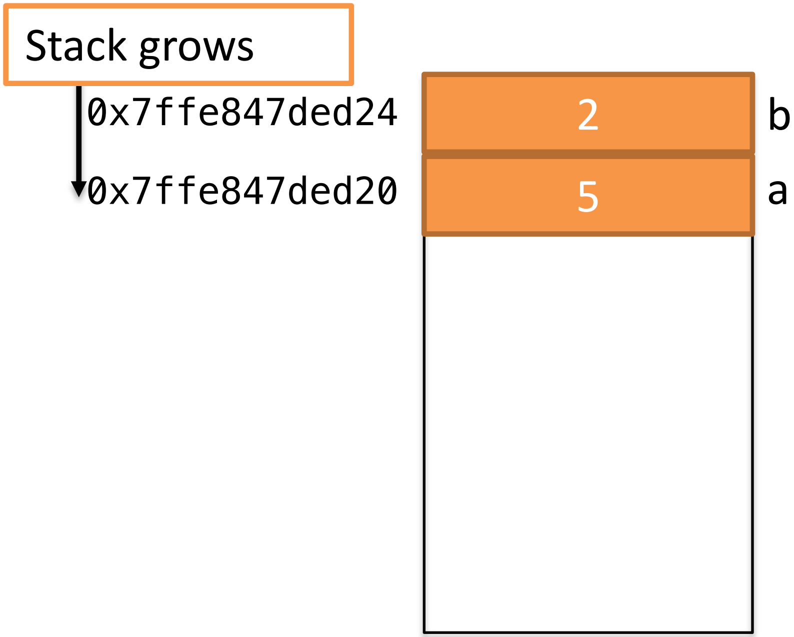
swap0.c

```c
void swap (int a, int b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
        (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
        a,b,temp);
    temp = a;
    a = b;
    b = temp;

    printf("\tswapped values: a=%d and b=%d\n",a,b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
        (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(a, b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
        (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

Stack popped execution returns to main

```
$ mygcc -o swap swap.c
$ ./swap
In main
addresses a=0x7ffe847ded20, b=0x7ffe847ded24
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffe847decfc, b=0x7ffe847decf8,
temp=0x7ffe847ded04
parameters values: a=5, b=2, temp=32766
swapped values: a=2 and b=5
```

Variables a and b have their original values, swap fails

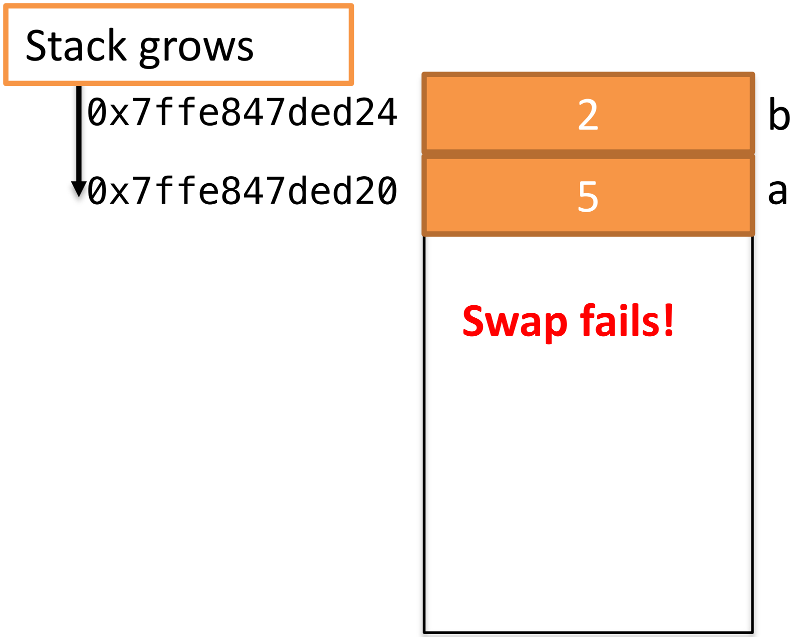


swap0.c

```
void swap (int a, int b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            a,b,temp);
    temp = a;
    a = b;
    b = temp;

    printf("\tswapped values: a=%d and b=%d\n",a,b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(a, b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

```
$ mygcc –o swap swap.c
$ ./swap
In main
addresses a=0x7ffe847ded20, b=0x7ffe847ded24
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffe847decfc, b=0x7ffe847decf8,
temp=0x7ffe847ded04
parameters values: a=5, b=2, temp=32766
swapped values: a=2 and b=5
Back in main
addresses a=0x7ffe847ded20, b=0x7ffe847ded24
values a=5 and b=2
```

Variables a and b have their original values, swap fails



swap0.c

```c
void swap (int a, int b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            a,b,temp);
    temp = a;
    a = b;
    b = temp;

    printf("\tswapped values: a=%d and b=%d\n",a,b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(a, b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

```
$ mygcc –o swap swap.c
$ ./swap
In main
addresses a=0x7ffe847ded20, b=0x7ffe847ded24
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffe847decfc, b=0x7ffe847decf8,
temp=0x7ffe847ded04
parameters values: a=5, b=2, temp=32766
swapped values: a=2 and b=5
Back in main
addresses a=0x7ffe847ded20, b=0x7ffe847ded24
values a=5 and b=2
```

Try passing address of a and b to func, then swap using pointers

swap1.c

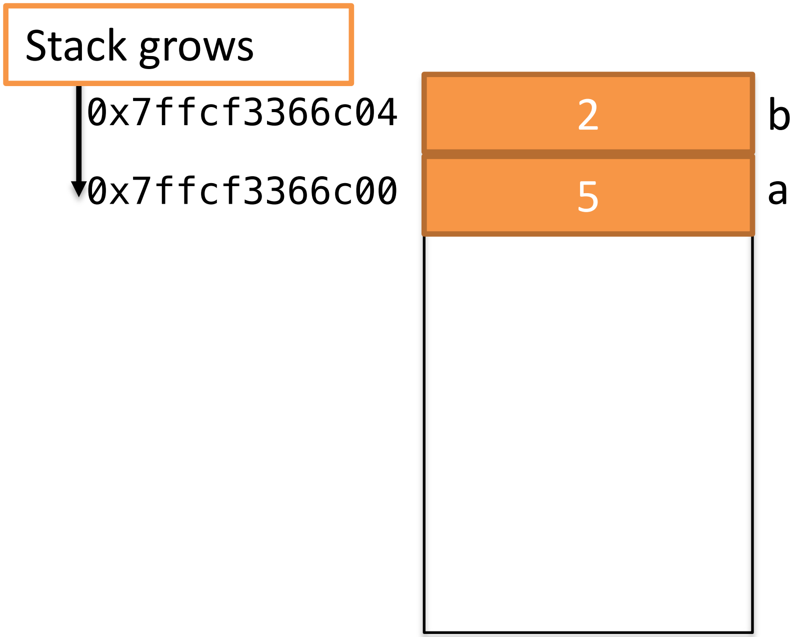
Use pointer to hold address of a and b

```c
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    a = b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

There is a subtle problem with this code, let's find it!

Pass address of a and b to func instead of value

Try passing address of a and b to func, then swap using pointers



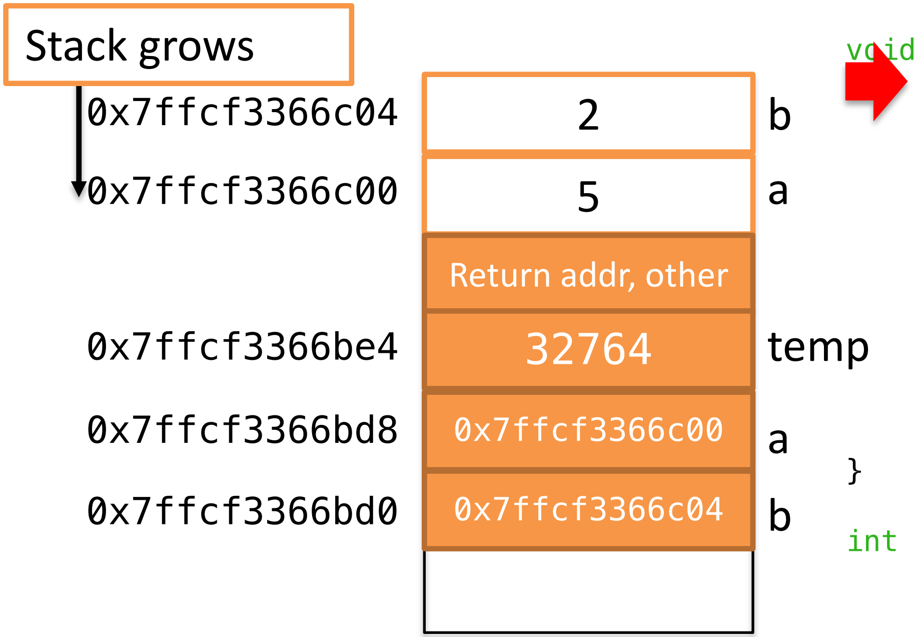
swap1.c

```
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    a = b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

Pass address of a and b by using &a and &b

```
$ mygcc -o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
```

Push address of a and b onto stack, also create local variable temp on stack



Stack grows

0x7ffcf3366c04 | 2 | b

0x7ffcf3366c00 | 5 | a

Return addr, other

0x7ffcf3366be4 | 32764 | temp

0x7ffcf3366bd8 | 0x7ffcf3366c00 | a

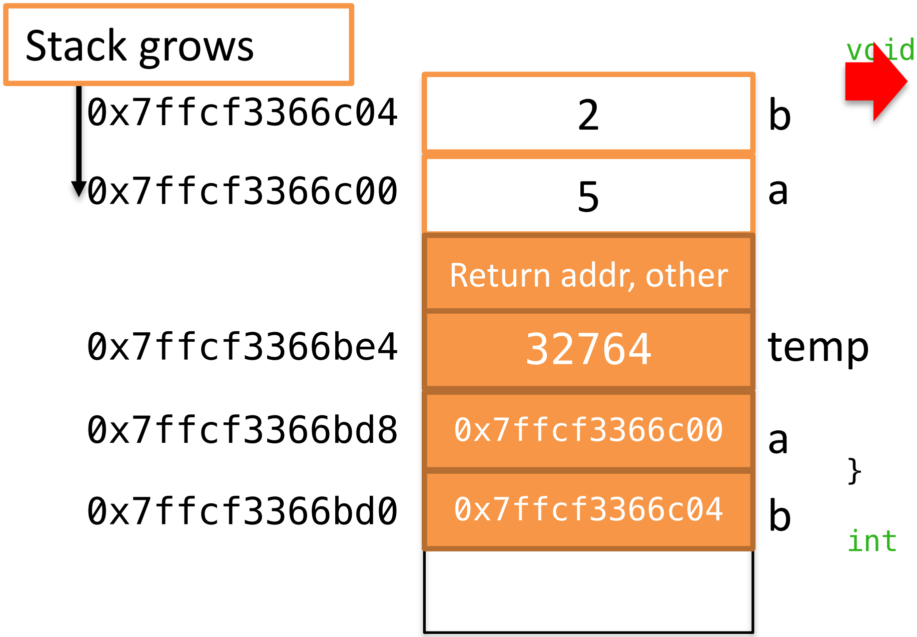
0x7ffcf3366bd0 | 0x7ffcf3366c04 | b

swap1.c

```
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    a = b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

```
$ mygcc -o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
```

Push address of a and b onto stack, also create local variable temp on stack



Stack grows

0x7ffcf3366c04	2	b
0x7ffcf3366c00	5	a
	Return addr, other	
0x7ffcf3366be4	32764	temp
0x7ffcf3366bd8	0x7ffcf3366c00	a
0x7ffcf3366bd0	0x7ffcf3366c04	b

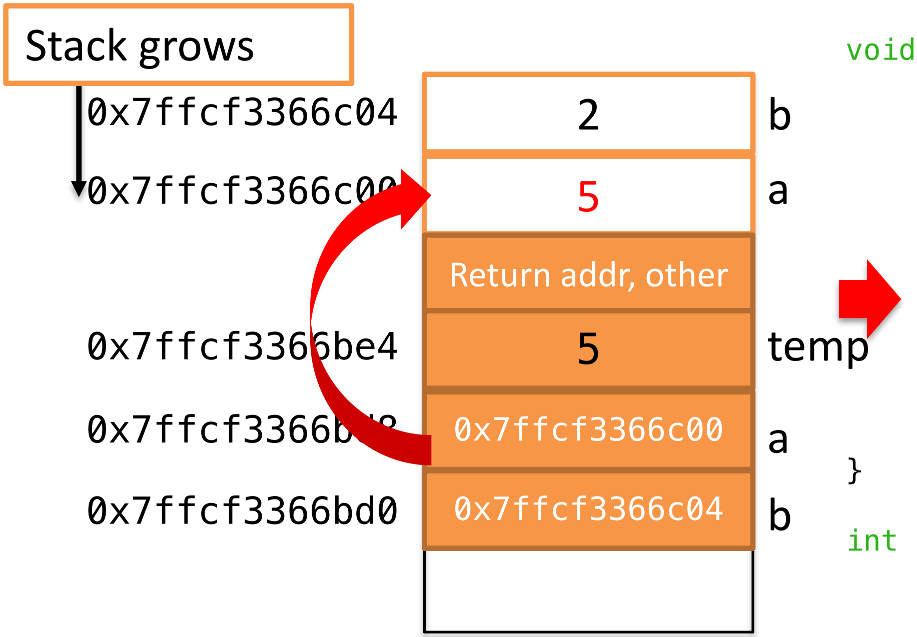
swap1.c

```
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    a = b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

Receive parameters as pointers, so they store the memory addresses passed

```
$ mygcc -o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
```

Set temp to value of a by derefencing with *a to get value



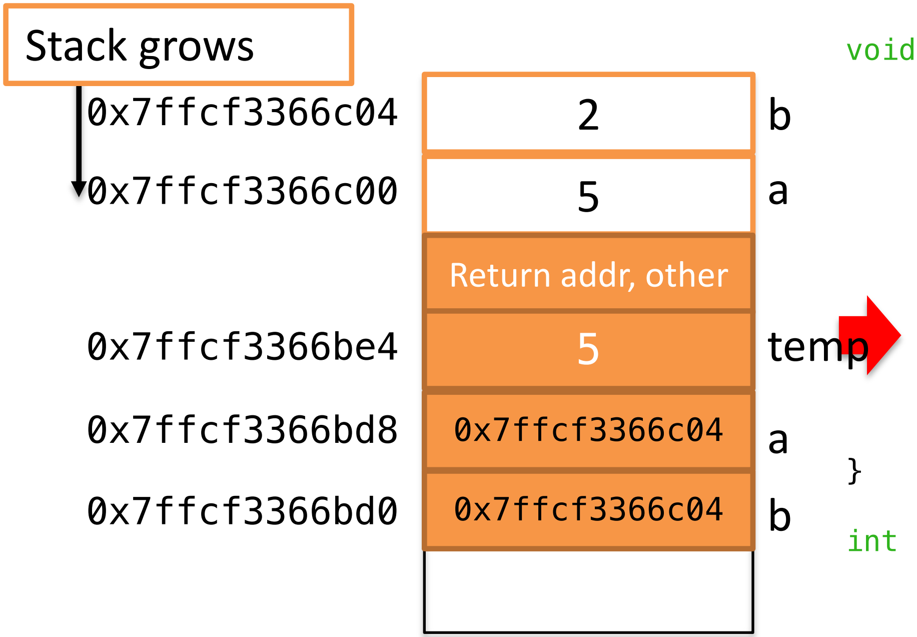
swap1.c

```
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    a = b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

Temp get value of a by deferencing with *a

```
$ mygcc –o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffcf3366bd8, b=0x7ffcf3366bd0,
temp=0x7ffcf3366be4
parameters values: a=5, b=2, temp=32764
```

Set a = b, is this what we want?



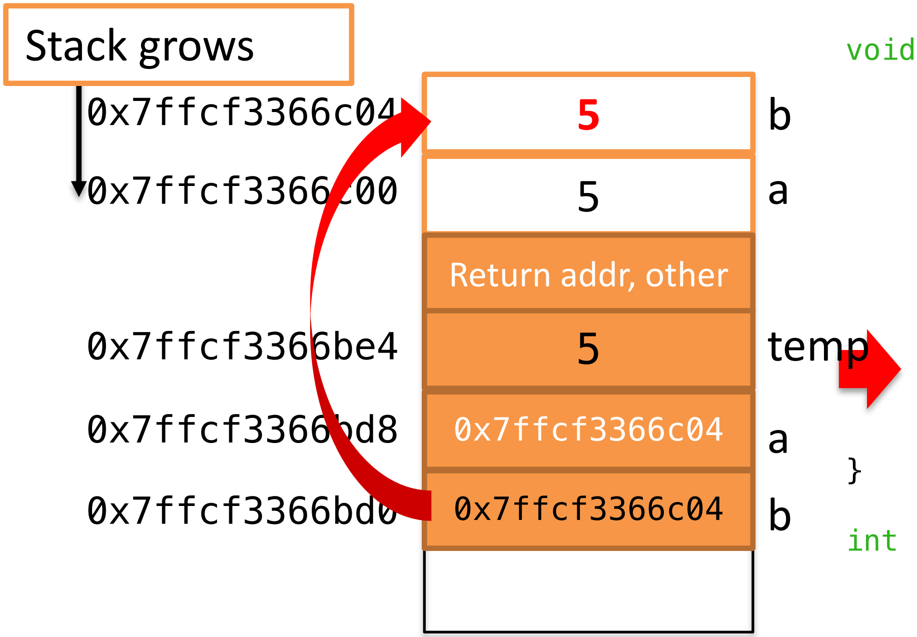
swap1.c

```
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    a = b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

a set equal b
Is this what we want?

```
$ mygcc -o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffcf3366bd8, b=0x7ffcf3366bd0,
temp=0x7ffcf3366be4
parameters values: a=5, b=2, temp=32764
```

Set b value to temp by using *b



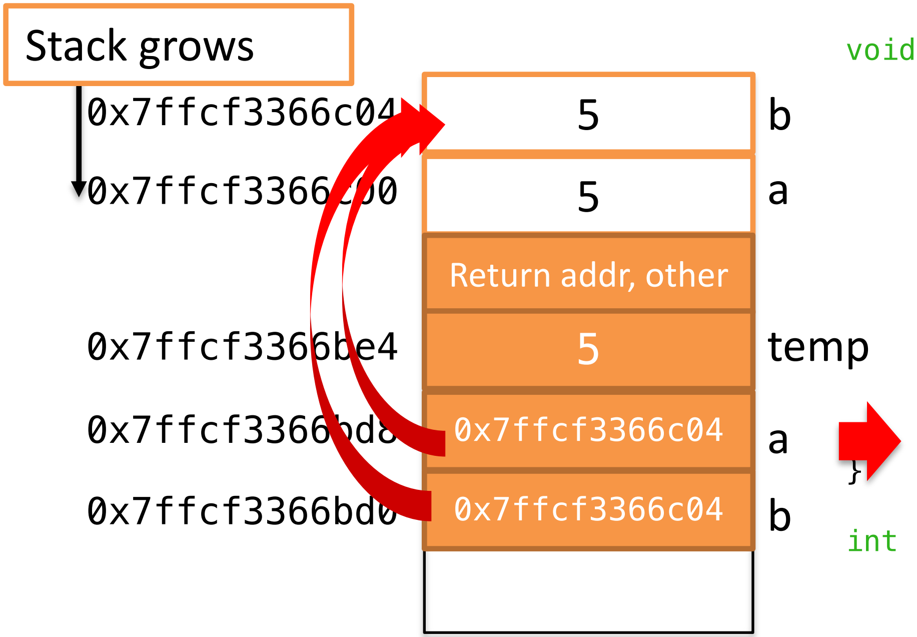
swap1.c

```
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    a = b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

Value of b set to temp (5)

```
$ mygcc -o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffcf3366bd8, b=0x7ffcf3366bd0,
temp=0x7ffcf3366be4
parameters values: a=5, b=2, temp=32764
```

Deference a and b to get values, looks like a problem!



swap1.c

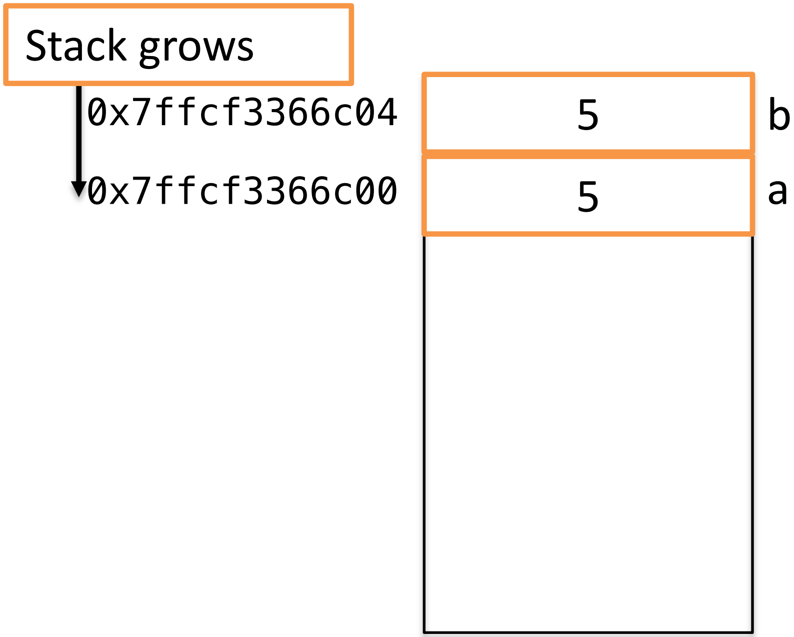
```
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    a = b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

Deference pointers to get values for a and b

```
$ mygcc -o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffcf3366bd8, b=0x7ffcf3366bd0,
temp=0x7ffcf3366be4
parameters values: a=5, b=2, temp=32764
swapped values: a=5 and b=5
```

Wait, what!?!?!?

Problem persists in main, what went wrong?


Stack grows

0x7ffcf3366c04 | 5 | b

0x7ffcf3366c00 | 5 | a

swap1.c

```
void swap (int *a, int *b) {
     int temp;
     printf("In swap, before making swap\n");
     printf("\taddresses: a=%p, b=%p, temp=%p\n",
             (void *)&a, (void *)&b, (void *)&temp);
     printf("\tparameters values: a=%d, b=%d, temp=%d\n",
             *a,*b,temp);
     temp = *a;
     a = b;
     *b = temp;

     printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
     int a = 5;
     int b = 2;
     printf("In main\n");
     printf("\taddresses a=%p, b=%p\n",
             (void *)&a, (void *)&b);
     printf("\tvalues a=%d and b=%d\n",a,b);

     printf("Calling swap function\n");
     swap(&a, &b);

     printf("Back in main\n");
     printf("\taddresses a=%p, b=%p\n",
             (void *)&a, (void *)&b);
     printf("\tvalues a=%d and b=%d\n",a,b);

     return 0;
}
```

What went wrong?

```
$ mygcc -o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffcf3366bd8, b=0x7ffcf3366bd0,
temp=0x7ffcf3366be4
parameters values: a=5, b=2, temp=32764
swapped values: a=5 and b=5
Back in main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=5
```

Problem persists in main, what went wrong?

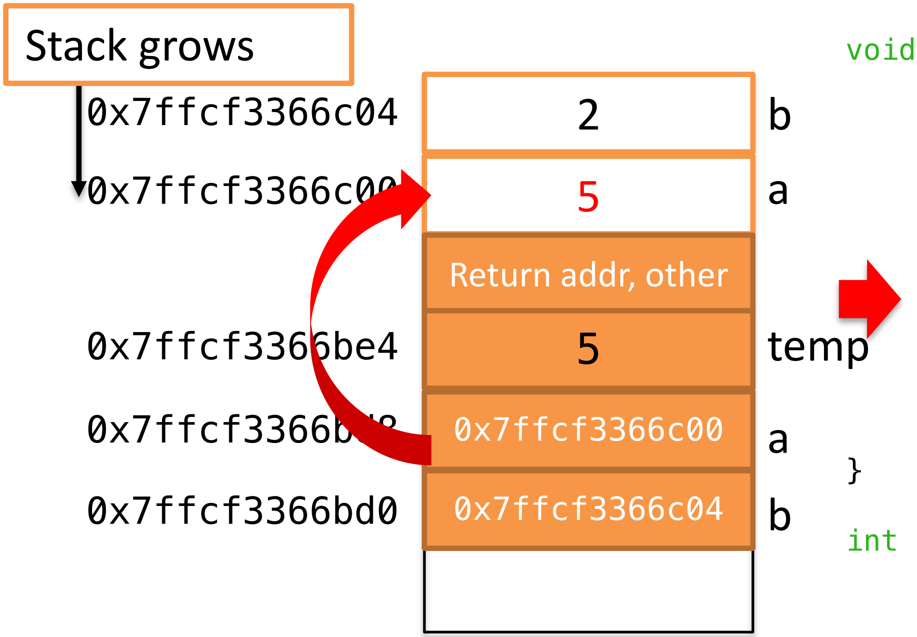
swap2.c

Set value of a to value of b with *a = *b

Previously set a = b so a got b’s memory address instead of its value!

```
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    *a = *b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

Set temp to value of a by derefencing with *a to get value



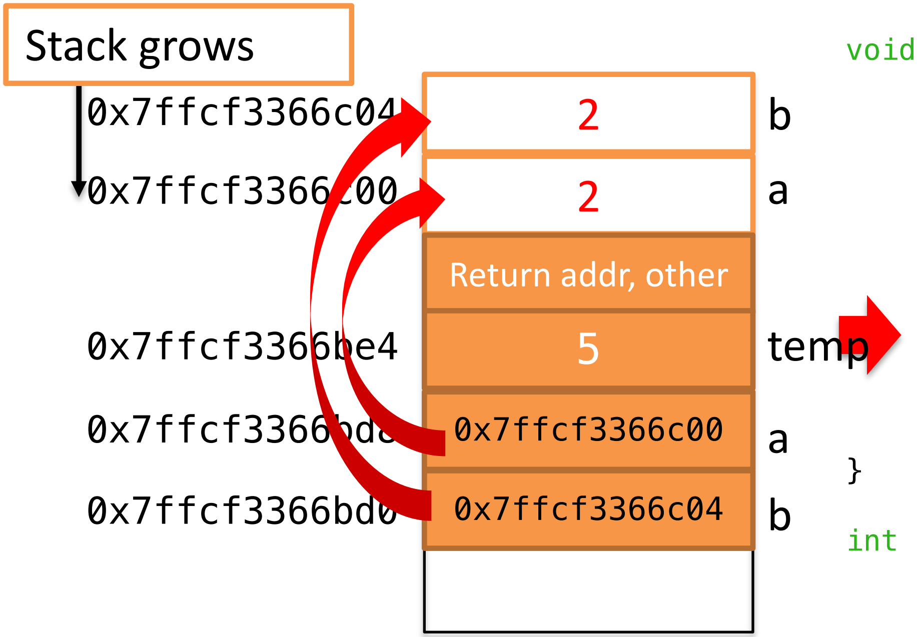
swap2.c

```
void swap (int *a, int *b) {
  int temp;
  printf("In swap, before making swap\n");
  printf("\taddresses: a=%p, b=%p, temp=%p\n",
        (void *)&a, (void *)&b, (void *)&temp);
  printf("\tparameters values: a=%d, b=%d, temp=%d\n",
        *a,*b,temp);
  temp = *a;
  *a = *b;
  *b = temp;

  printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
  int a = 5;
  int b = 2;
  printf("In main\n");
  printf("\taddresses a=%p, b=%p\n",
        (void *)&a, (void *)&b);
  printf("\tvalues a=%d and b=%d\n",a,b);

  printf("Calling swap function\n");
  swap(&a, &b);

  printf("Back in main\n");
  printf("\taddresses a=%p, b=%p\n",
        (void *)&a, (void *)&b);
  printf("\tvalues a=%d and b=%d\n",a,b);

  return 0;
}
```

Temp get value of a by deferencing with *a

```
$ mygcc -o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffcf3366bd8, b=0x7ffcf3366bd0,
temp=0x7ffcf3366be4
parameters values: a=5, b=2, temp=32764
```

Set a = b, is this what we want?



Stack grows

0x7ffcf3366c04 2 b

0x7ffcf3366c00 2 a

Return addr, other

0x7ffcf3366be4 5 temp

0x7ffcf3366bd8 0x7ffcf3366c00 a

0x7ffcf3366bd0 0x7ffcf3366c04 b

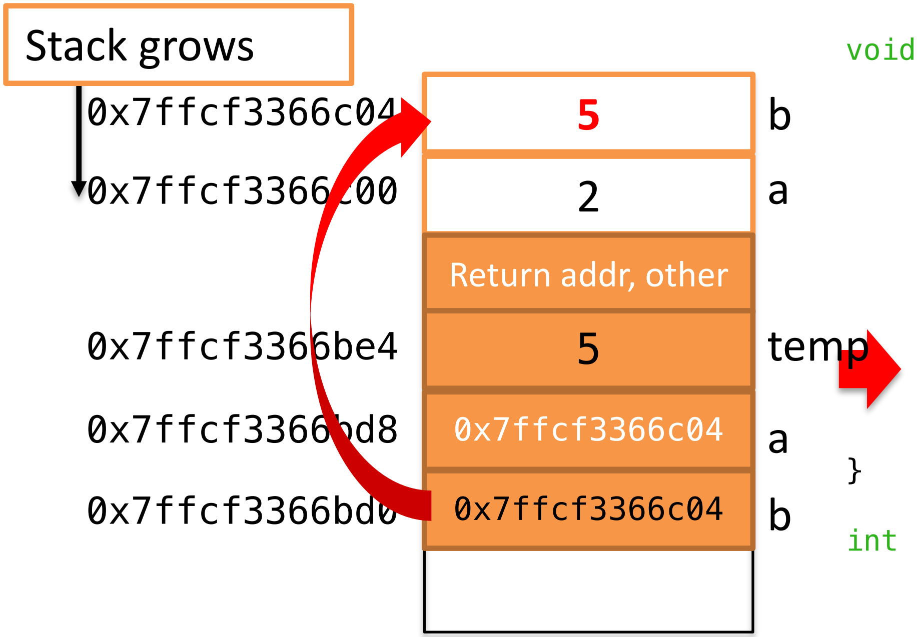
swap1.c

```
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    *a = *b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

Set value of a = value b by deferencing

```
$ mygcc –o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffcf3366bd8, b=0x7ffcf3366bd0,
temp=0x7ffcf3366be4
parameters values: a=5, b=2, temp=32764
```

Set b value to temp by using *b



Stack grows

0x7ffcf3366c04	5	b
0x7ffcf3366c00	2	a
	Return addr, other	
0x7ffcf3366be4	5	temp
0x7ffcf3366bd8	0x7ffcf3366c04	a
0x7ffcf3366bd0	0x7ffcf3366c04	b

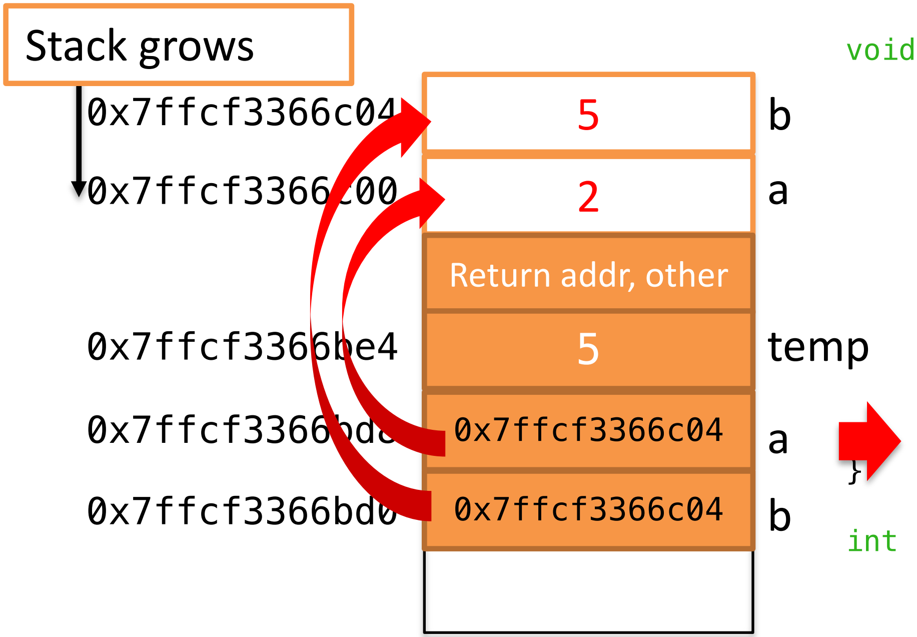
swap1.c

```
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    *a = *b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

Value of b set to temp (5)

```
$ mygcc -o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffcf3366bd8, b=0x7ffcf3366bd0,
temp=0x7ffcf3366be4
parameters values: a=5, b=2, temp=32764
```

Deference a and b to get values, looks like it worked!



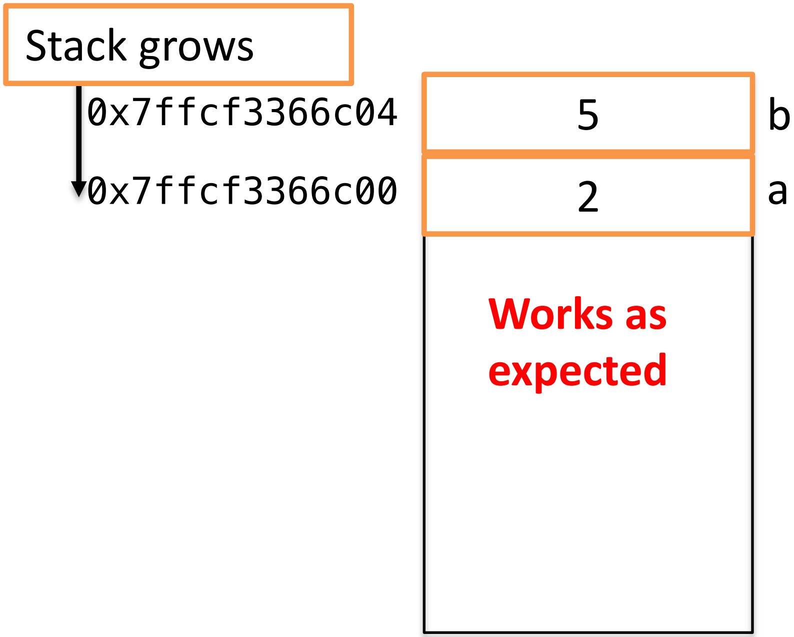
swap2.c

```
void swap (int *a, int *b) {
  int temp;
  printf("In swap, before making swap\n");
  printf("\taddresses: a=%p, b=%p, temp=%p\n",
        (void *)&a, (void *)&b, (void *)&temp);
  printf("\tparameters values: a=%d, b=%d, temp=%d\n",
        *a,*b,temp);
  temp = *a;
  *a = *b;
  *b = temp;

  printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
  int a = 5;
  int b = 2;
  printf("In main\n");
  printf("\taddresses a=%p, b=%p\n",
        (void *)&a, (void *)&b);
  printf("\tvalues a=%d and b=%d\n",a,b);

  printf("Calling swap function\n");
  swap(&a, &b);

  printf("Back in main\n");
  printf("\taddresses a=%p, b=%p\n",
        (void *)&a, (void *)&b);
  printf("\tvalues a=%d and b=%d\n",a,b);

  return 0;
}
```

Deference pointers to get values for a and b

```
$ mygcc -o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffcf3366bd8, b=0x7ffcf3366bd0,
temp=0x7ffcf3366be4
parameters values: a=5, b=2, temp=32764
swapped values: a=2 and b=5
```

Returning to main, all is well, swap worked!



Works as expected

swap1.c

```
void swap (int *a, int *b) {
    int temp;
    printf("In swap, before making swap\n");
    printf("\taddresses: a=%p, b=%p, temp=%p\n",
            (void *)&a, (void *)&b, (void *)&temp);
    printf("\tparameters values: a=%d, b=%d, temp=%d\n",
            *a,*b,temp);
    temp = *a;
    a = b;
    *b = temp;

    printf("\tswapped values: a=%d and b=%d\n",*a,*b);
}

int main() {
    int a = 5;
    int b = 2;
    printf("In main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    printf("Calling swap function\n");
    swap(&a, &b);

    printf("Back in main\n");
    printf("\taddresses a=%p, b=%p\n",
            (void *)&a, (void *)&b);
    printf("\tvalues a=%d and b=%d\n",a,b);

    return 0;
}
```

```
$ mygcc –o swap swap1.c
$ ./swap
In main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=5 and b=2
Calling swap function
In swap, before making swap
addresses: a=0x7ffcf3366bd8, b=0x7ffcf3366bd0,
temp=0x7ffcf3366be4
parameters values: a=5, b=2, temp=32764
swapped values: a=2 and b=5
Back in main
addresses a=0x7ffcf3366c00, b=0x7ffcf3366c04
values a=2 and b=5
```

Agenda

1. You’ve seen the *idea* of pointers in Java

2. C pointers

3. Pass by value

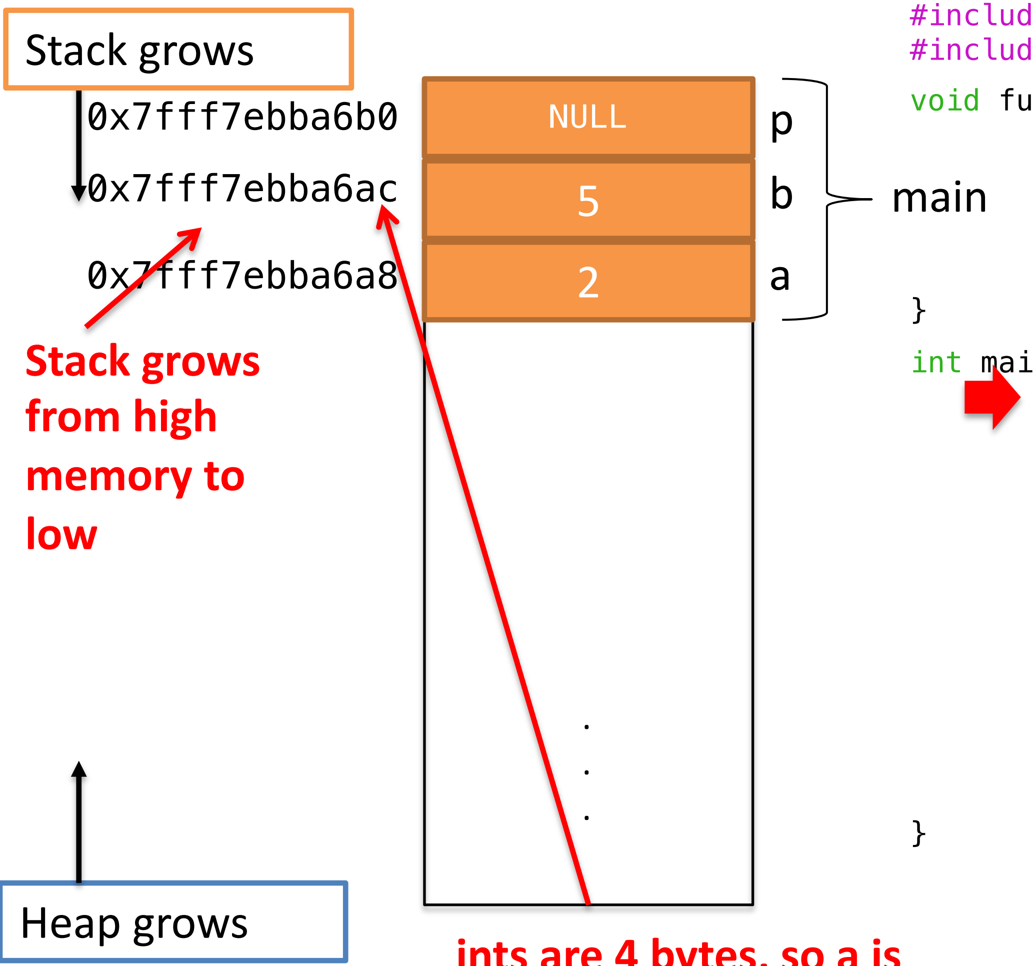
➡ 4. Activity

Stack stores local variables, grows downward

ptr_test.c

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

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

Stack stores local variables, grows downward



Stack grows from high memory to low

ptr_test.c

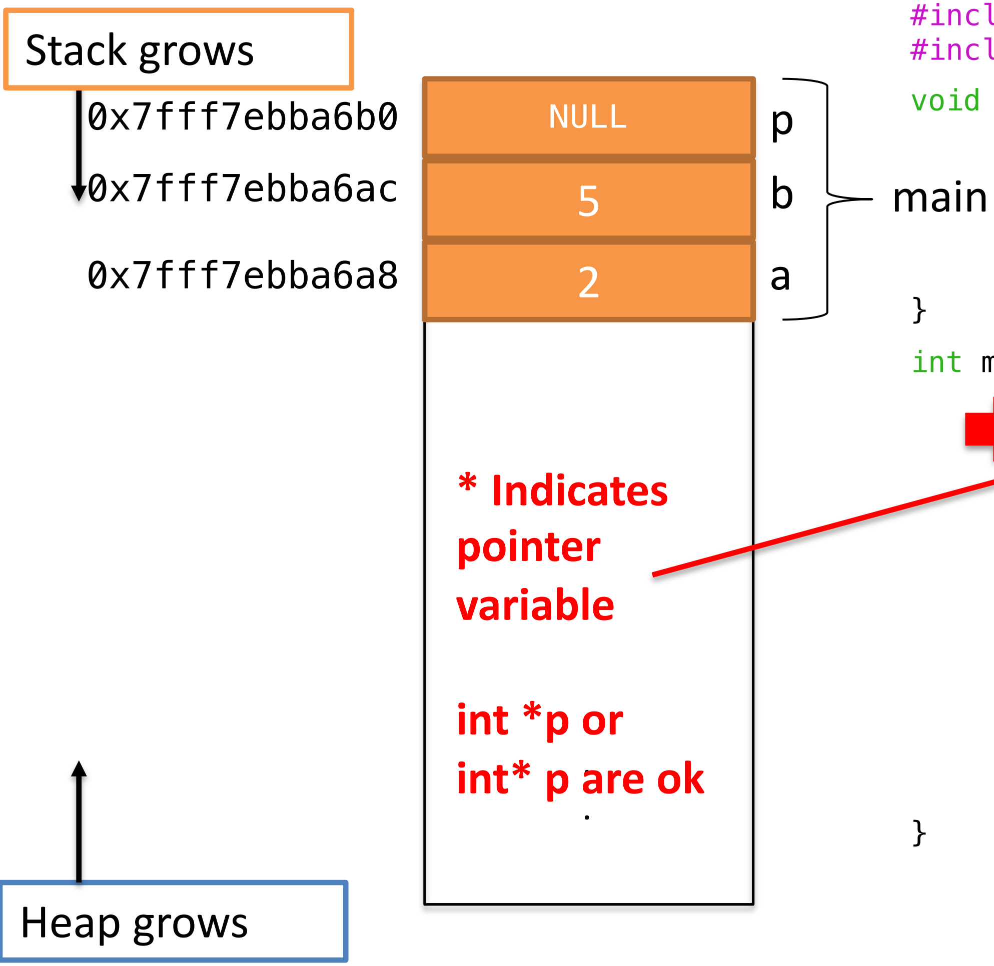
```
#include<stdio.h>
#include<stdlib.h>

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

Local variables stored on stack

ints are 4 bytes, so a is 4 bytes lower in memory than b

0x…6a8 + 4 = 0x…6ac

Pointers store memory address, not values like “normal” variables



*** Indicates pointer variable**

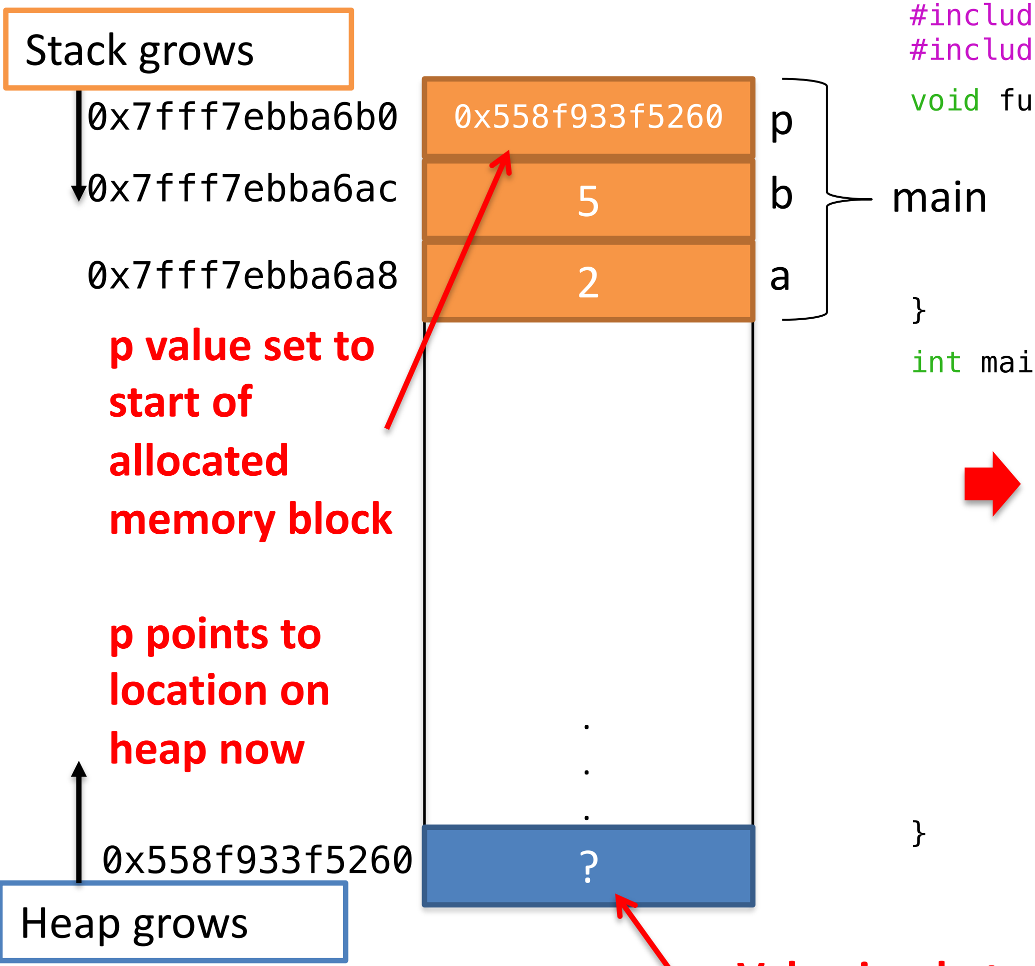
int *p or int* p are ok

ptr_test.c

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

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;
        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

Creates entry on stack filled with NULL

Malloc allocates memory on heap and returns pointer to start of block allocated



p value set to start of allocated memory block

p points to location on heap now

malloc in stdlib.h

ptr_test.c

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

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

malloc allocates 4 bytes on heap

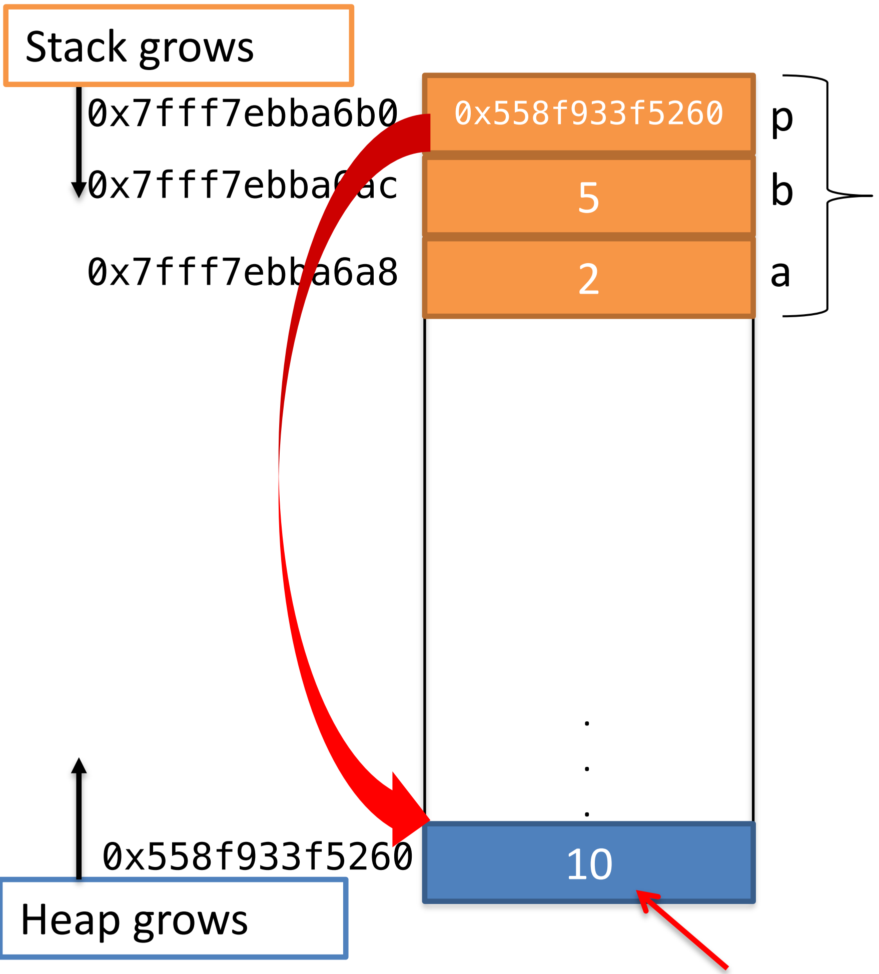
Malloc returns start of memory allocated as void pointer

Cast as integer pointer

Value is whatever was left in memory

malloc is like *new* in Java
Does not have to be freed in Java

Deference pointers using *



Stack grows

0x7fff7ebba6b0

0x7fff7ebba6ac

0x7fff7ebba6a8

0x558f933f5260

p

5

b

2

a

main

0x558f933f5260

10

Heap grows

ptr_test.c

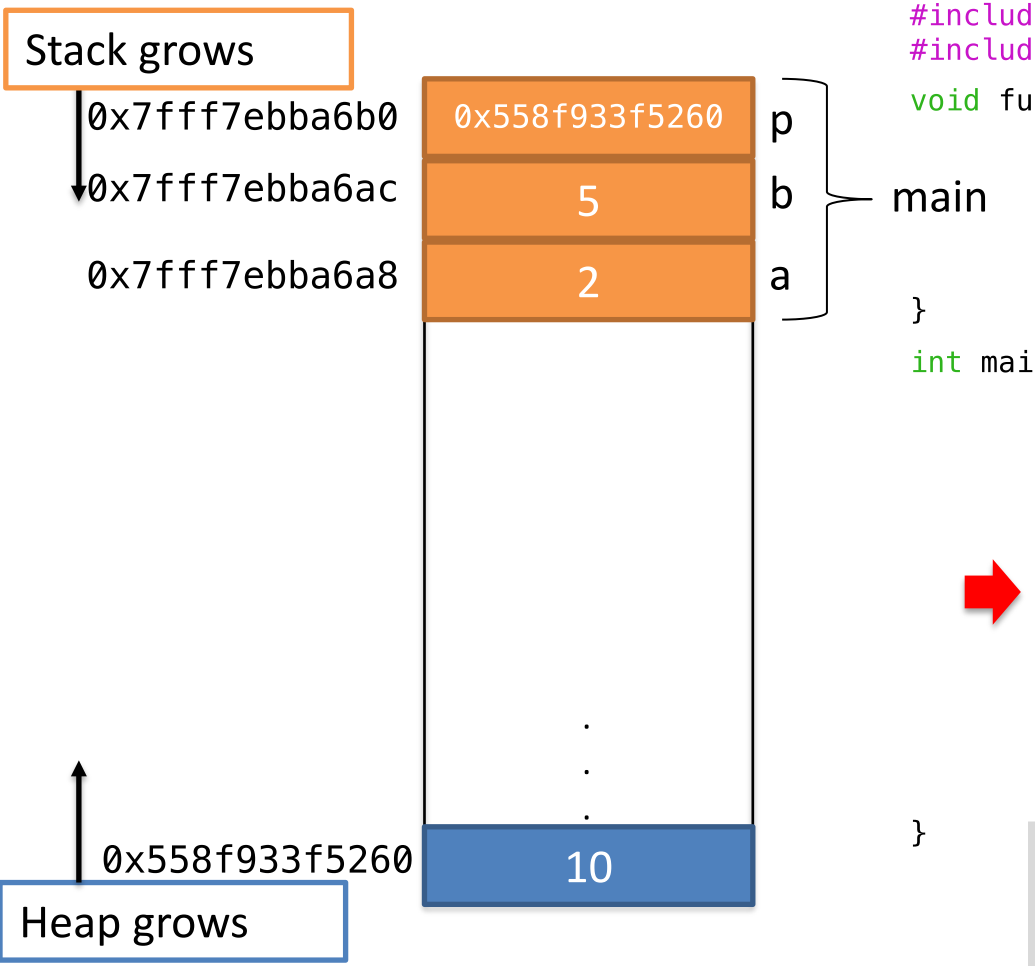
```c
#include<stdio.h>
#include<stdlib.h>

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

*** deferences pointer**

Go to where p points and set value to 10

Value on heap now 10

& gives the address of a variable



ptr_test.c

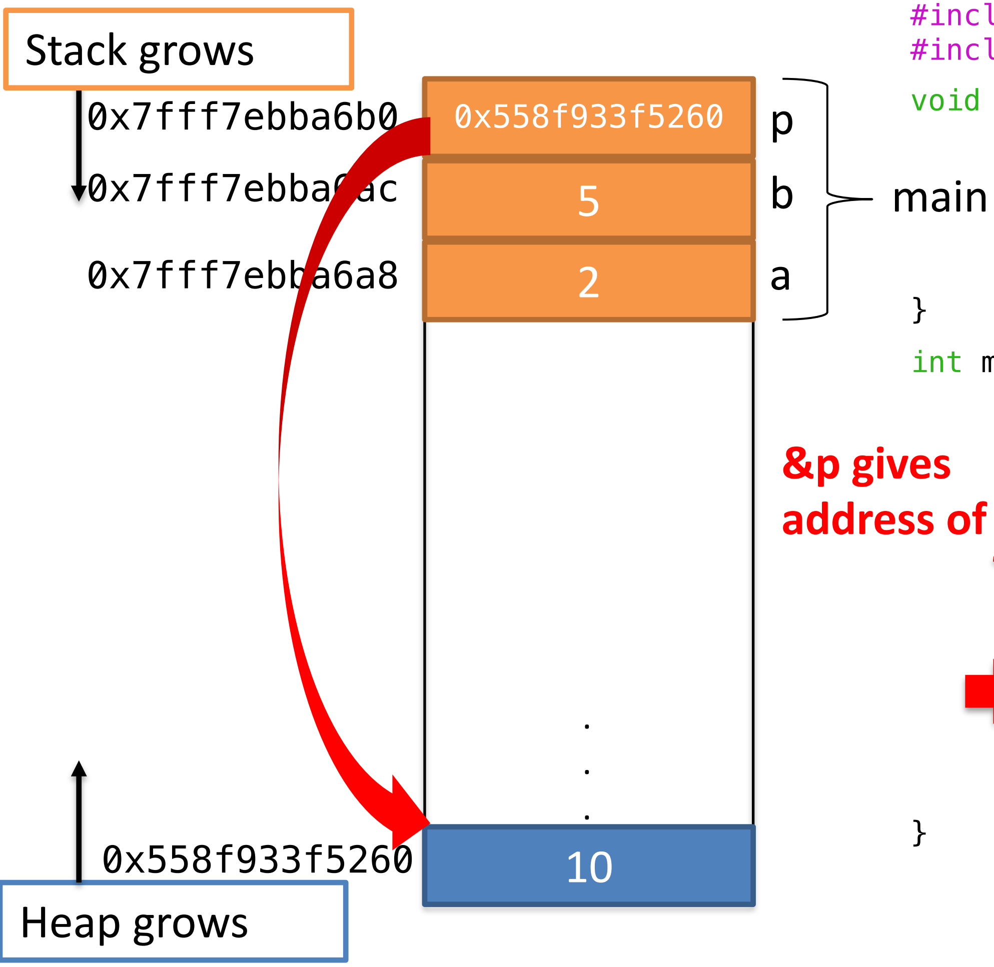
```
#include<stdio.h>
#include<stdlib.h>

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

Cast address to void pointer

& gives “address of” variable

```
$ mygcc -o ptr_test ptr_test.c
$ ./ptr_test
a: value 2 at 0x7fff7ebba6a8
b: value 5 at 0x7fff7ebba6ac
```

Use &, *, and parameter, to get address of variable, variable value, and heap address



ptr_test.c

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

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

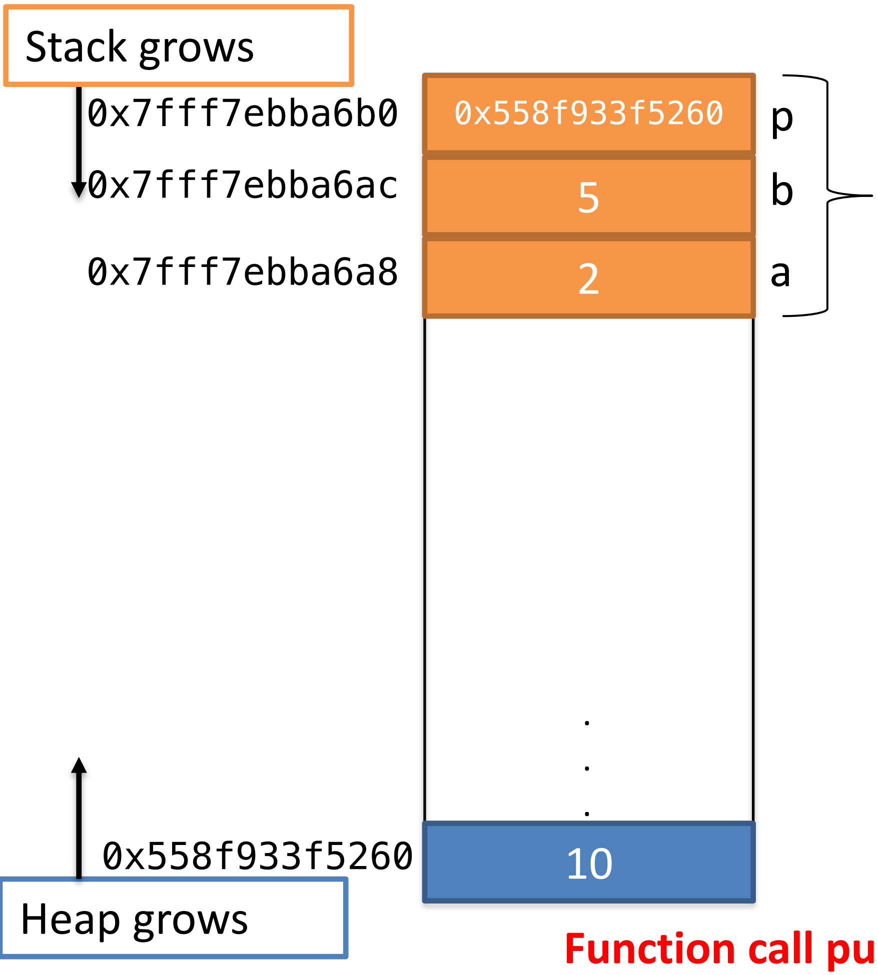
&p gives address of p

p's value is heap address

***p deferences p to get value (10)**

```
$ mygcc -o ptr_test ptr_test.c
$ ./ptr_test
a: value 2 at 0x7fff7ebba6a8
b: value 5 at 0x7fff7ebba6ac
p is at 0x7fff7ebba6b0
p: value 10 at 0x558f933f5260
```

Function call pushes return address, local variables and parameters on stack



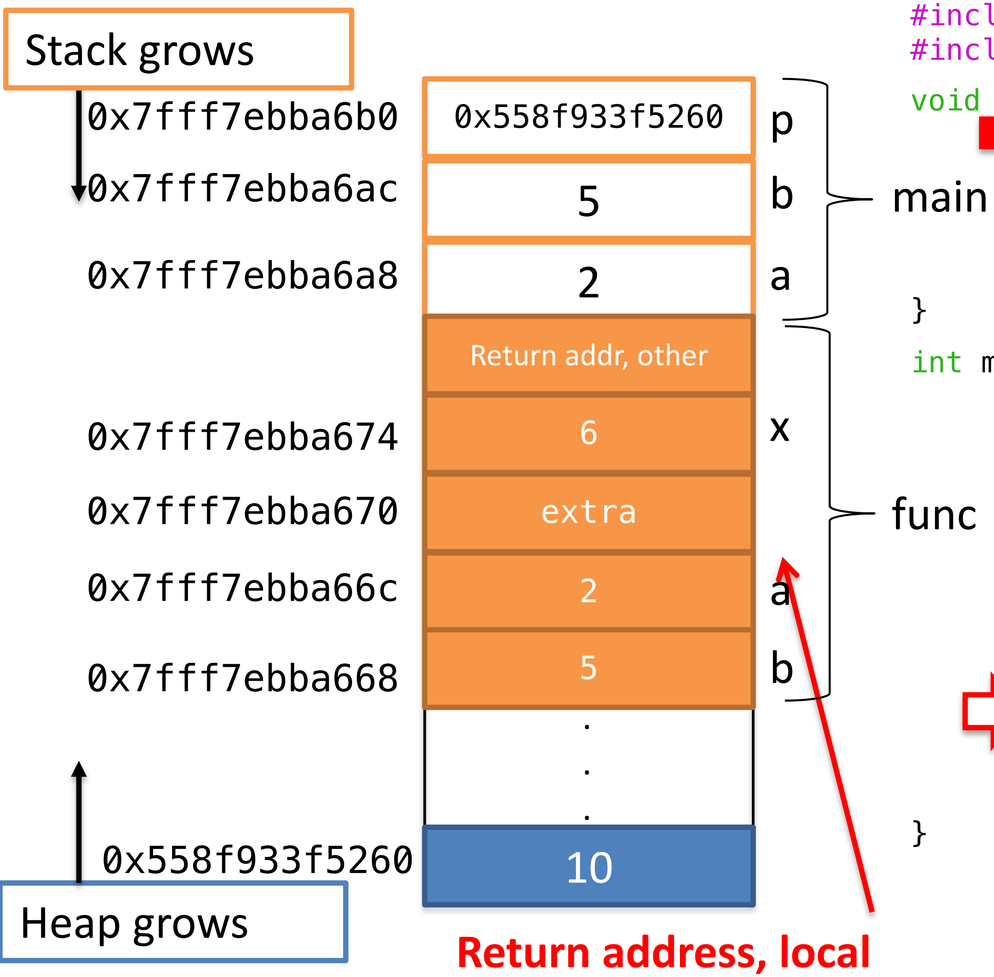
ptr_test.c

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

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

```
$ mygcc -o ptr_test ptr_test.c
$ ./ptr_test
a: value 2 at 0x7fff7ebba6a8
b: value 5 at 0x7fff7ebba6ac
p is at 0x7fff7ebba6b0
p: value 10 at 0x558f933f5260
```

Function call pushes return address, parameters, and local variables onto stack

Function call pushes return address, local variables and parameters on stack



Return address, local variables, and parameters pushed onto stack

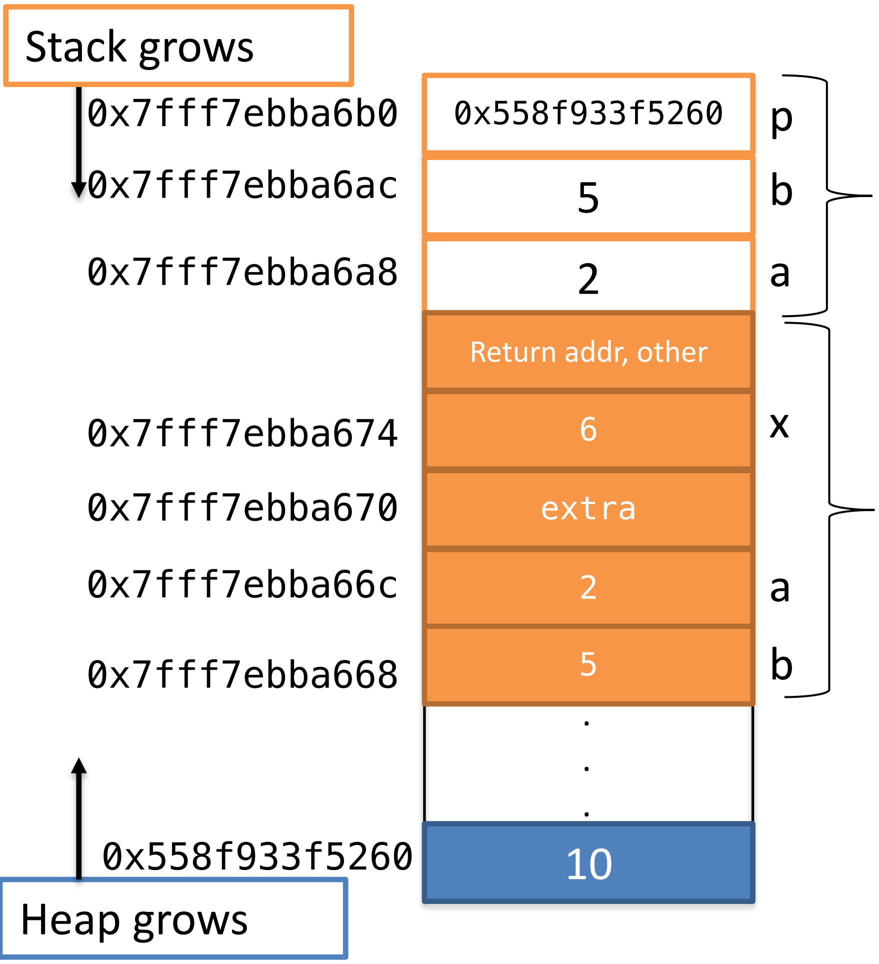
ptr_test.c

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

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

Return address is this command

```
$ mygcc -o ptr_test ptr_test.c
$ ./ptr_test
a: value 2 at 0x7fff7ebba6a8
b: value 5 at 0x7fff7ebba6ac
p is at 0x7fff7ebba6b0
p: value 10 at 0x558f933f5260
```

Function call pushes return address, local variables and parameters on stack

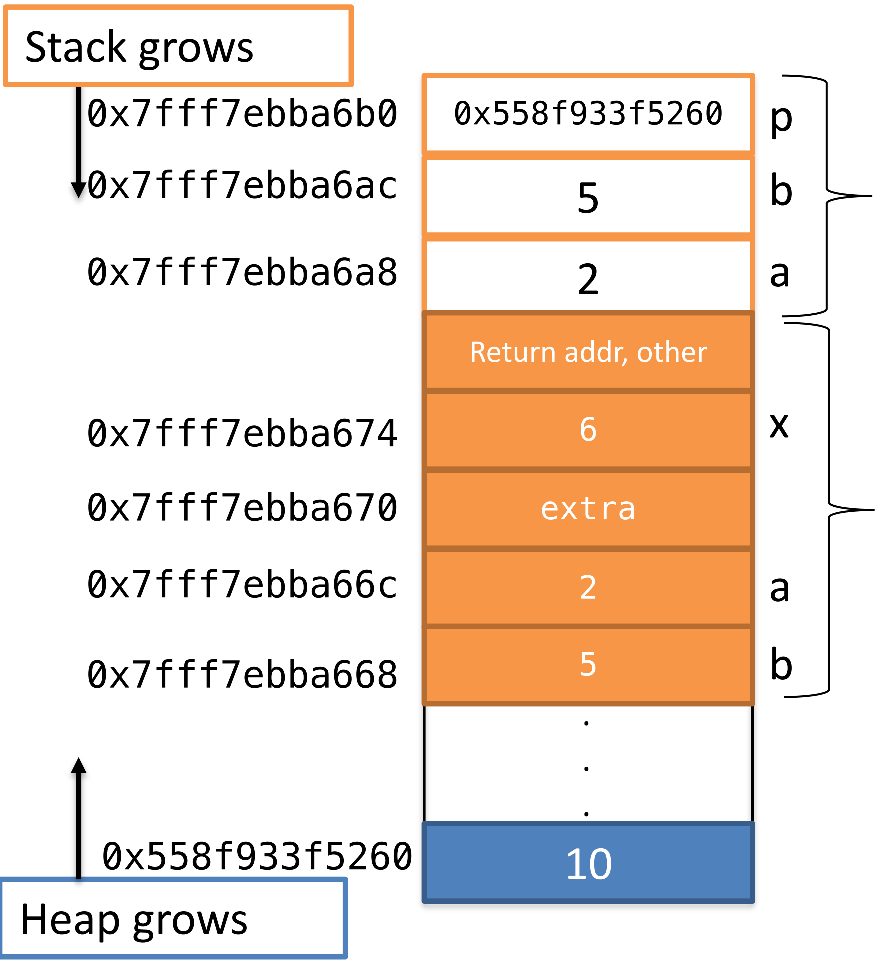


ptr_test.c

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

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

```
$ mygcc –o ptr_test ptr_test.c
$ ./ptr_test
a: value 2 at 0x7fff7ebba6a8
b: value 5 at 0x7fff7ebba6ac
p is at 0x7fff7ebba6b0
p: value 10 at 0x558f933f5260
In func
a: value 2 at 0x7fff7ebba66c
b: value 5 at 0x7fff7ebba668
x: value 6 at 0x7fff7ebba674
```

Function call pushes return address, local variables and parameters on stack



Stack grows

0x7fff7ebba6b0	0x558f933f5260	p
0x7fff7ebba6ac	5	b
0x7fff7ebba6a8	2	a
	Return addr, other	
0x7fff7ebba674	6	x
0x7fff7ebba670	extra	
0x7fff7ebba66c	2	a
0x7fff7ebba668	5	b
	. . .	
0x558f933f5260	10	

Heap grows

main

func

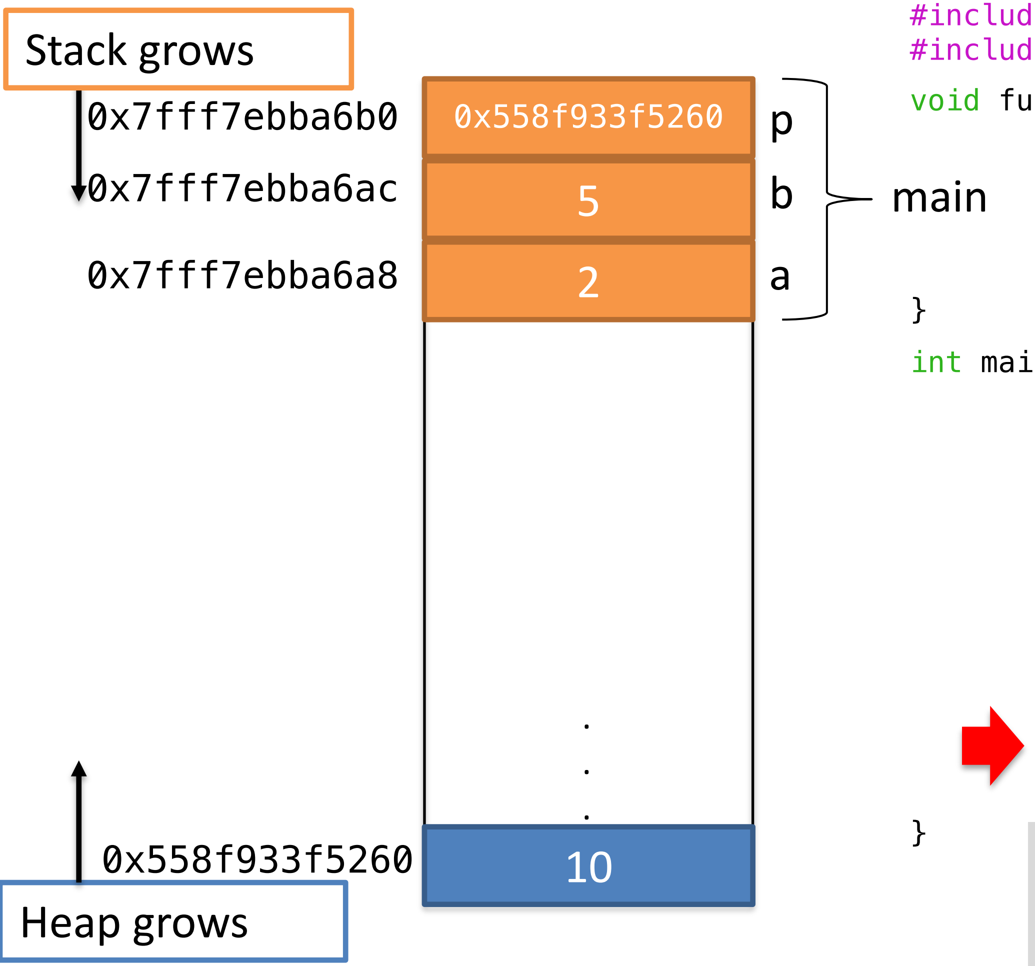
ptr_test.c

func ends
Pop stack

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

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

```
$ mygcc -o ptr_test ptr_test.c
$ ./ptr_test
a: value 2 at 0x7fff7ebba6a8
b: value 5 at 0x7fff7ebba6ac
p is at 0x7fff7ebba6b0
p: value 10 at 0x558f933f5260
In func
a: value 2 at 0x7fff7ebba66c
b: value 5 at 0x7fff7ebba668
x: value 6 at 0x7fff7ebba674
```

When function ends, pop stack to remove local variables, return address, parameters



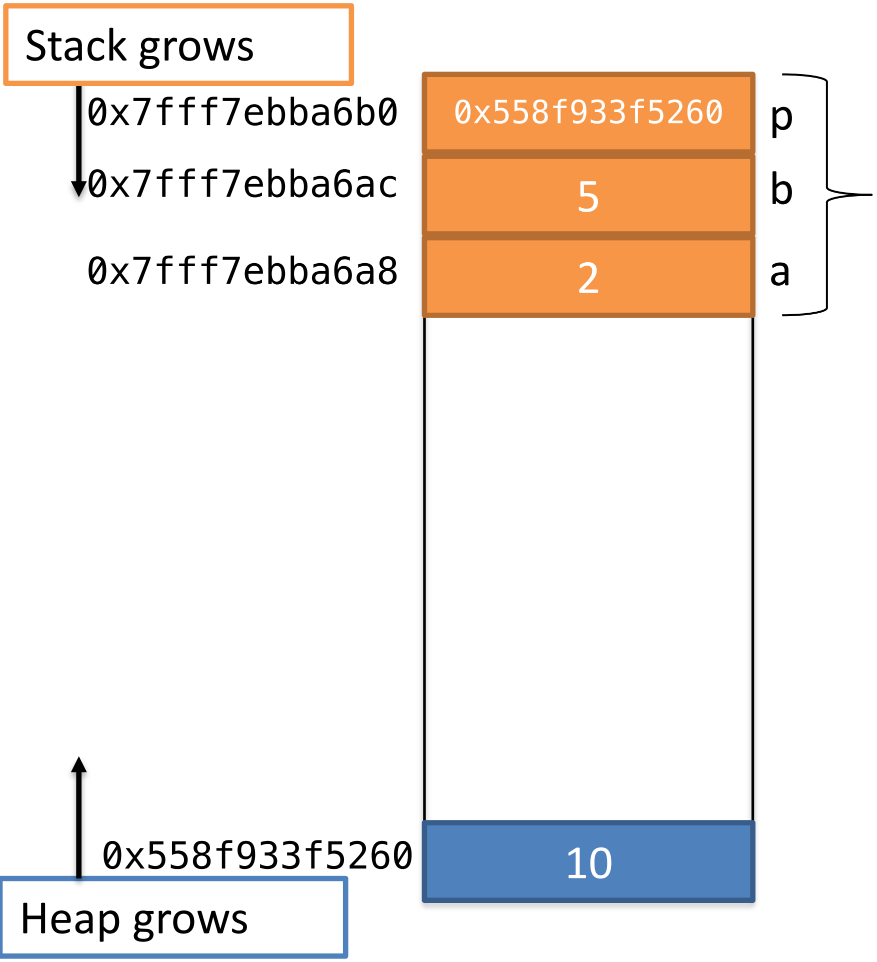
ptr_test.c

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

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

Resume execution at return address

```
$ mygcc -o ptr_test ptr_test.c
$ ./ptr_test
a: value 2 at 0x7fff7ebba6a8
b: value 5 at 0x7fff7ebba6ac
p is at 0x7fff7ebba6b0
p: value 10 at 0x558f933f5260
In func
a: value 2 at 0x7fff7ebba66c
b: value 5 at 0x7fff7ebba668
x: value 6 at 0x7fff7ebba674
```

Do not forget to free malloc'd memory!



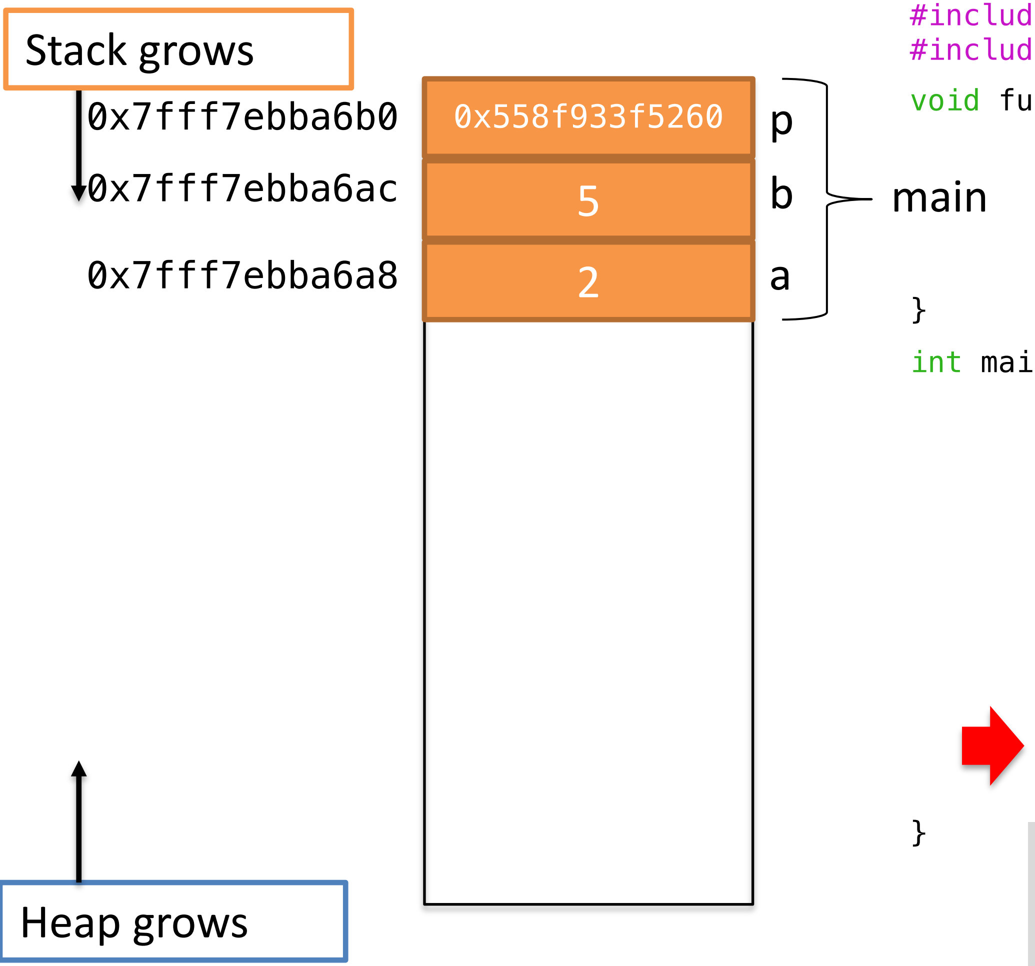
ptr_test.c

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

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

Free memory on heap
Otherwise, memory leak!

```
$ mygcc -o ptr_test ptr_test.c
$ ./ptr_test
a: value 2 at 0x7fff7ebba6a8
b: value 5 at 0x7fff7ebba6ac
p is at 0x7fff7ebba6b0
p: value 10 at 0x558f933f5260
In func
a: value 2 at 0x7fff7ebba66c
b: value 5 at 0x7fff7ebba668
x: value 6 at 0x7fff7ebba674
```

Do not forget to free malloc'd memory!



ptr_test.c

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

void func(int a, int b) {
        int x = 6;
        printf("In func\n");
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("x: value %d at %p\n",x,(void *)&x);
}

int main(int argc, char *argv[]) {
        int a = 2, b = 5;
        int *p = NULL;

        p = (int *) malloc(sizeof(int));
        *p = 10;
        printf("a: value %d at %p\n",a,(void *)&a);
        printf("b: value %d at %p\n",b,(void *)&b);
        printf("p is at %p\n",(void *)&p);
        printf("p: value %d at %p\n",*p,(void *)p);

        func(a,b);
        free(p);

        return 0;
}
```

Free memory on heap

Otherwise, memory leak!

```
$ mygcc -o ptr_test ptr_test.c
$ ./ptr_test
a: value 2 at 0x7fff7ebba6a8
b: value 5 at 0x7fff7ebba6ac
p is at 0x7fff7ebba6b0
p: value 10 at 0x558f933f5260
In func
a: value 2 at 0x7fff7ebba66c
b: value 5 at 0x7fff7ebba668
x: value 6 at 0x7fff7ebba674
```