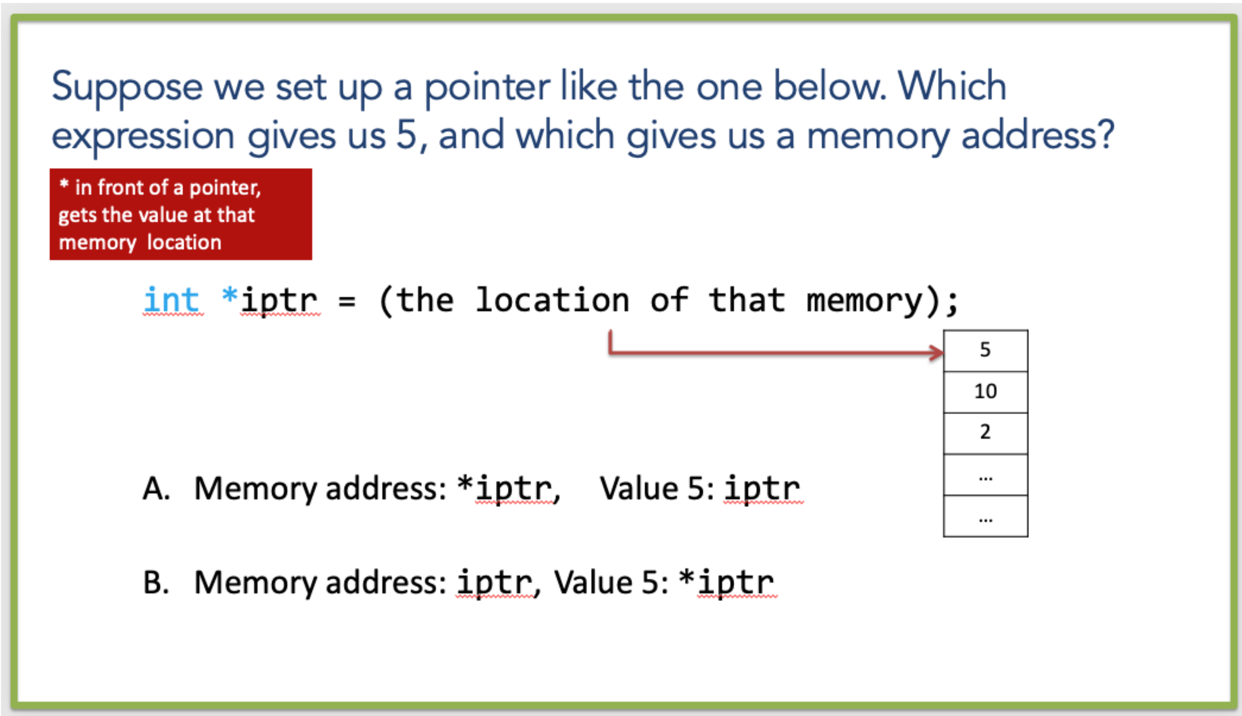
CS31 Worksheet: Week 5: Pointers & Memory Management

Q1.



Q2. Declare pointers to the following variables:

```
int main(void){
   float y = 10;
   double z = 20.2;


  return 0;
}
```

If we can declare variables on the stack, why do we need to dynamically allocate things on the heap?

A. There is more space available on the heap.

B. Heap memory is better. (Why?)

C. We may not know a variable's size in advance.

D. The stack grows and shrinks automatically.

E. Some other reason.

If we can declare varia
do we need to dynami
the heap?

Which region would we expect the PC register (program counter) to point to?

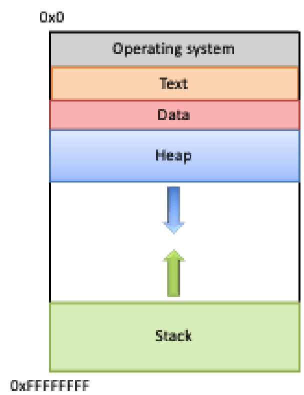
A. OS

B. Text

C. Data

D. Heap

E. Stack



What should happen if we try to access an address that's NOT in one of these regions?

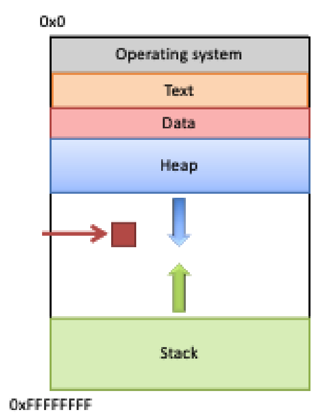
A. The address is allocated to your program.

B. The OS warns your program.

C. The OS kills your program.

D. The access fails, try the next instruction.

E. Something else

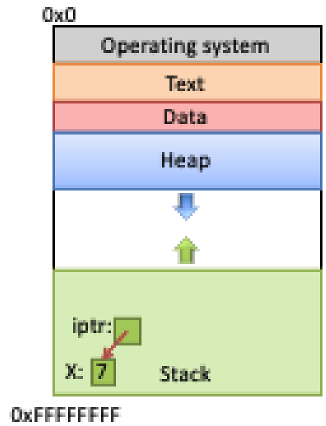


What would this print?

```c
int main(void) {
  int x = 7;
  int *iptr  = &x;
  int *iptr2 = &x;

  printf("%d %d ", x, *iptr);
  *iptr2 = 5;
  printf("%d %d ", x, *iptr);

  return 0;
}
```



A. 7 7 7 7   B. 7 7 7 5   C. 7 7 5 5   D. Something else

Given these two setup statements, how many of the following dereference operations are <u>invalid</u>?

Setup:

```c
int *ptr = &x;      // ptr stores address of int x
char *chptr = &ch;  // chptr stores address of char ch
```

Dereference operations:

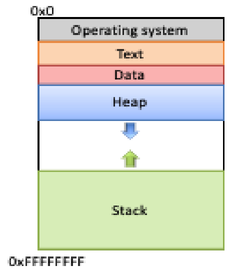
```c
1) *ptr = 6;
2) *chptr = 'a';
3) int y = *ptr + 4;
4) ptr = NULL, *ptr = 6;
```

A: 1   B: 2   C: 3   D: 4

What will this do?

```
int main(void) {
  int *ptr;
  printf("%d", *ptr);
}
```

A. Print 0
B. Print a garbage value
C. Segmentation fault
D. Something else



Takeaway: If you're not immediately assigning it something when you declare it, initialize your pointers to NULL.

You're designing a system. What should happen if a program requests memory and the system doesn't have enough available?

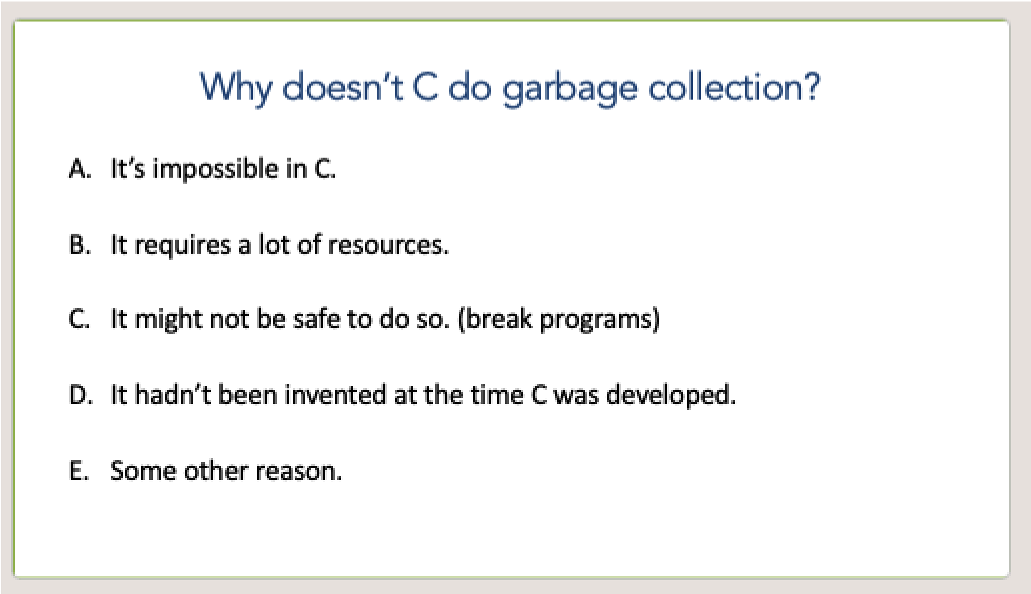
A. The OS kills the requesting program.
B. The OS kills another program to make room.
C. malloc gives it as much memory as is available.
D. malloc returns NULL.
E. Something else.

What do you expect to happen to the 100-byte chunk if we do this?

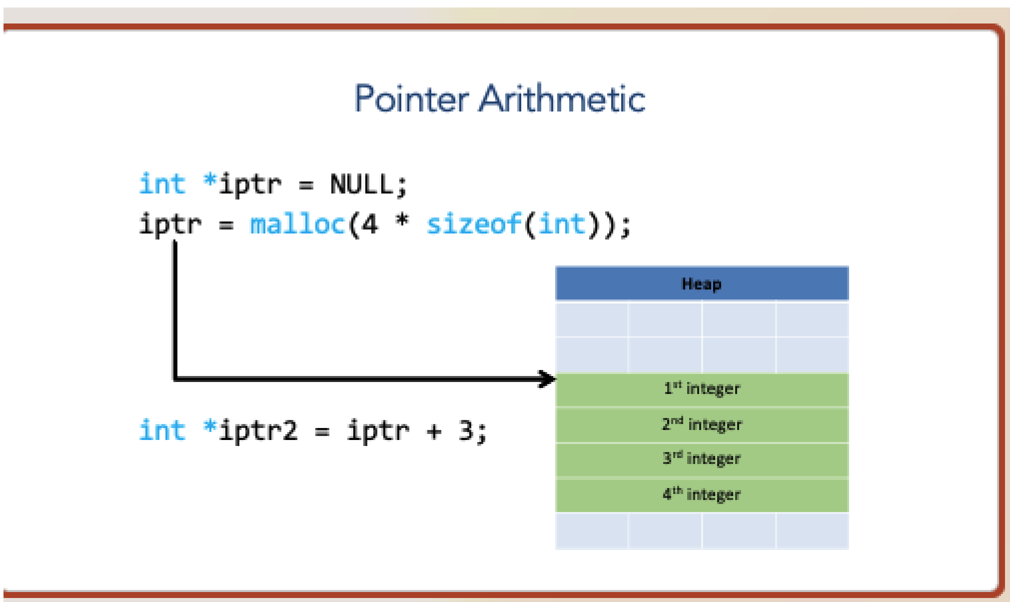
```
// What happens to these 100 bytes?
int *ptr = malloc(100);

ptr = malloc(2000);
```

A. The 100-byte chunk will be lost.
B. The 100-byte chunk will be automatically freed (garbage collected) by the OS.
C. The 100-byte chunk will be automatically freed (garbage collected) by C.
D. The 100-byte chunk will be the first 100 bytes of the 2000-byte chunk.
E. The 100-byte chunk will be added to the 2000-byte chunk (2100 bytes total).



What is the value of iptr2 below?

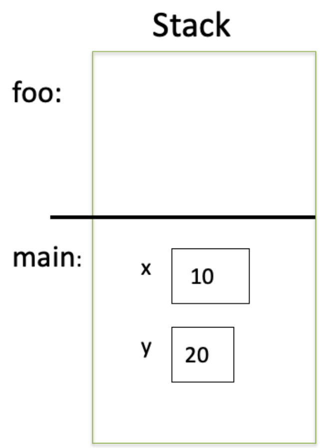


Draw out the stack frame for the following C code snippets:

Pass by Pointer - Example

```c
int main(void){
  int x, y;
  x = 10; y = 20;
  foo(&x, y);
  …
}

void foo(int *b, int c){
  c = 99
 *b = 8;  // Stack drawn here
}
```



Passing Arrays

- An array argument's value is its base address
- Array parameter "points to" its array argument

```c
int main(void){
  int array[10];
  foo(array, 10);
}
void foo(int arr[], int n){
  arr[2] = 6;
}
```

array base address

What's an alternative way to pass the array from foo to main?

What’s wrong with the following code assuming main calls copy_array? Draw out the stack diagram **after** copy_array executes to see the error.