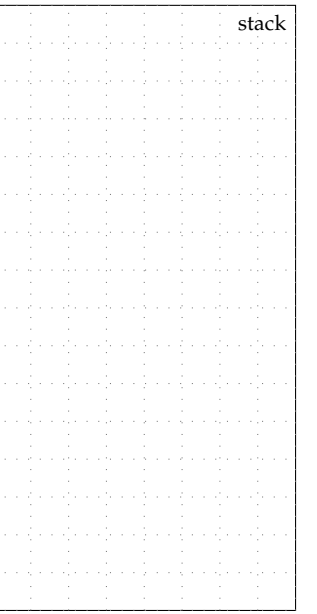
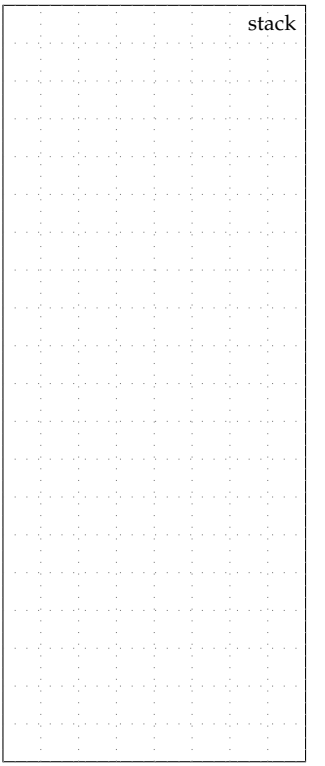
Draw a state diagram corresponding to the execution of this program

```
01 void avg( int* result_ptr,
02            int x, int y )
03 {
04   int sum = x + y;
05   *result_ptr = sum / 2;
06 }
07
08 int main( void )
09 {
10   int a = 10;
11   int b = 20;
12   int c;
13   avg( &c, a, b );
14   return 0;
15 }
```



zyBooks The course zyBook includes a coding lab to implement an in-place square function that uses call-by-pointer.

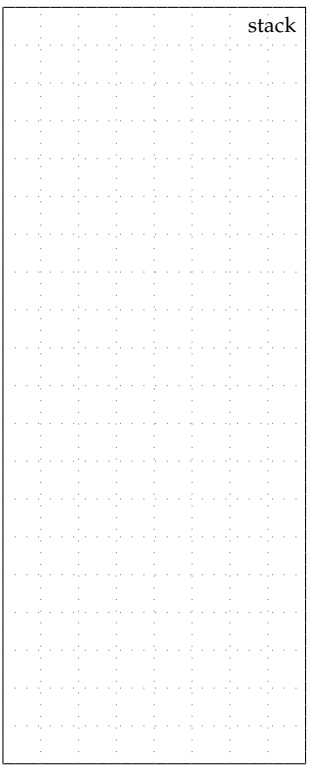
```c
#include <stddef.h>

typedef struct _node_t
{
  int              value;
  struct _node_t* next_p;
}
node_t;

int main( void )
{
  node_t node0;
  node0.value  = 3;
  node0.next_p = NULL;

  node_t node1;
  node1.value  = 4;
  node1.next_p = &node0;

  node_t node2;
  node2.value  = 5;
  node2.next_p = &node1;

  int sum = 0;
  node_t* curr_p = &node2;
  while ( curr_p != NULL ) {
    sum += curr_p->value;
    curr_p = curr_p->next_p;
  }
  return 0;
}
```



zyBooks The course zyBook includes a coding lab to implement a function to find the maximum value in a chain of nodes.

Draw a state diagram corresponding to the execution of this program

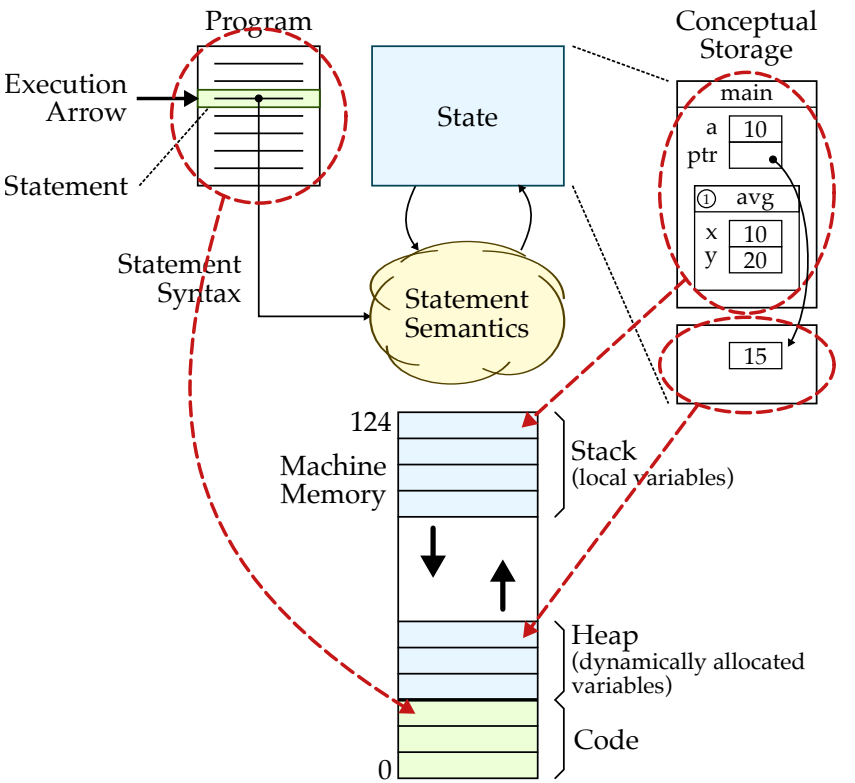
```
01 int strlen( char* str )
02 {
03   int i = 0;
04   while ( str[i] != '\0' )
05     i++;
06   return i;
07 }
08
09 int main( void )
10 {
11   char a[] = "ece2400";
12   int b = strlen( a );
13   return 0;
14 }
```



zyBooks The course zyBook includes a coding lab to implement a function to copy a string from a source array to a destination array.

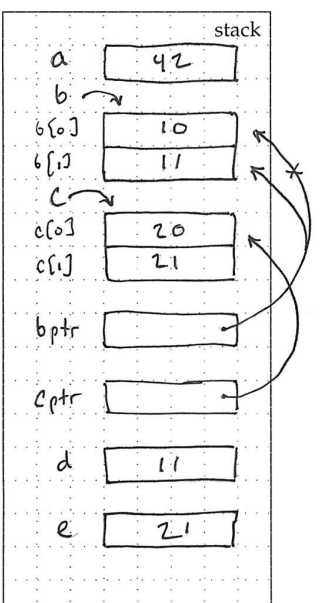
3. Mapping Conceptual Storage to Machine Memory

- Recall that our current use of state diagrams is conceptual
- Real machine uses memory to store variables
- Real machine does not use "arrows", uses memory addresses
- Heap is stored above code and grows *up*

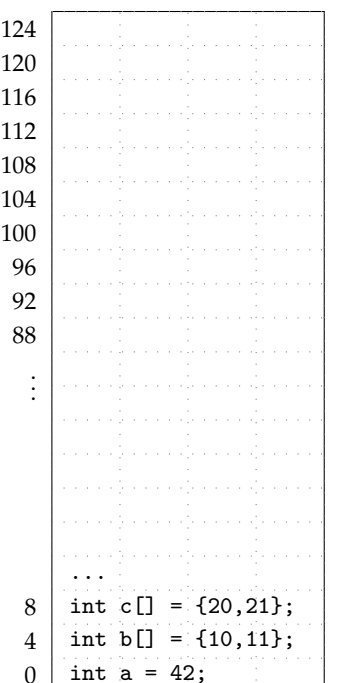


Draw both a conceptual storage and machine memory state diagram corresponding to the execution of this program

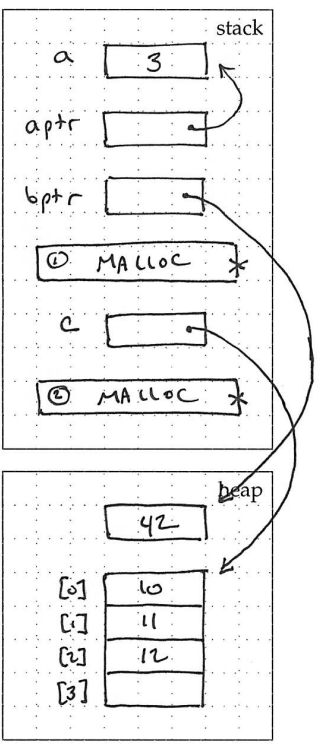
```
01 int a = 42;
02
03 int b[] = { 10, 11 };
04 int c[] = { 20, 21 };
05
06 int* b_ptr = b;
07 int* c_ptr = c;
08
09 b_ptr = b_ptr + 1;
10
11 int d = *b_ptr;
12 int e = c_ptr[1];
```



Memory
(4B word addr)



```
01 int  a = 3;
02 int* a_ptr = &a;
03
04 int* b_ptr = malloc( sizeof(int) );
05 *b_ptr = 42;
06
07 int* c = malloc( 4 * sizeof(int) );
08 c[0] = 10;
09 c[1] = 11;
10 c[2] = 12;
```



Memory
(4B word addr)