Login name _____ **Quiz 3** **Name** _____

CSE 131B

Signature _____ **Spring 2005** **Student ID** _____

1. Given the array declaration

C

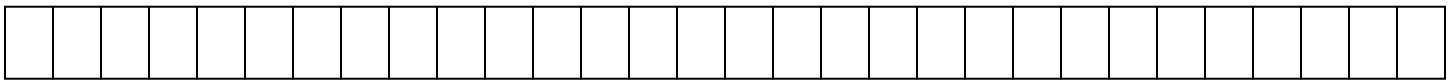
```
int a[7];
```

Oberon-like

```
VAR a : ARRAY 7 OF INTEGER;
```

Mark with an **A** the memory locations where we would find

```
      a[4]
a:
```



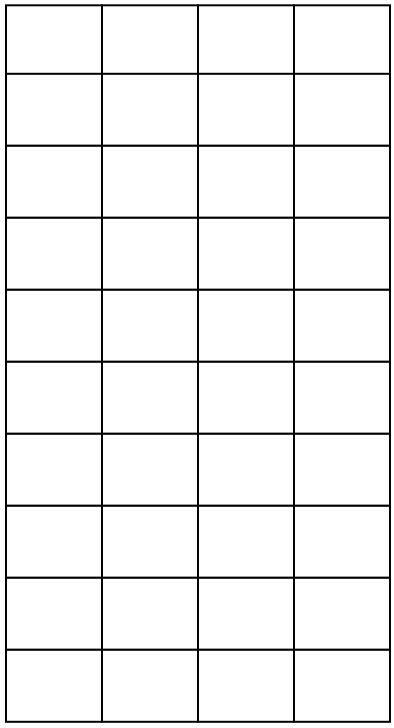
```
low memory                                                    high memory
```

2. Show the memory layout of the following C struct/record definition taking into consideration the **SPARC** data type memory alignment restrictions discussed in class. Fill bytes in memory with the appropriate struct/record member/field name. For example, if member/field name `p` takes 4 bytes, you will have 4 `p`'s in the appropriate memory locations. If the member/field is an array, use the name followed by the index number. For example, some number of `p0`s, `p1`s, `p2`s, etc. Place an `X` in any bytes of padding. Structs and unions are padded so the total size is evenly divisible by the most strict alignment requirement of its members.

```
struct foo {
   short  a;
   char   b[3];
   double c[2];
   char   d;
   short  e;
   char   f;
};

struct foo fubar;
```

```
fubar:
```



```
low memory

high memory
```

What is the `offsetof( struct foo, e )`? _____

What is the `sizeof( struct foo )`? _____

If `struct foo` had been defined as `union foo` instead, what would be the `sizeof( union foo )`? _____

3. Some languages (like C++) allow the programmer to define local block-scoped variables anywhere in a block and that variable name is scoped to that block from that point on to the end of the block. Needless to say, this complicates the compiler's scoping (intra-block local scope STO inserts) and type checking mechanisms and the ability to perform certain code improvements.

Consider the following valid C++ program fragment:

```
int main( char *argv[], int argc )
{
  int i = 2;

  while ( i == 2 )
  {
    i++;
    double i = 2.2;
    i++;
    if ( i > 2 )
    {
      cout << i << " "; // Output the current value of i followed by a space
      i--;
      char i = '2';
      i--;
      cout << i << " "; // Output the current value of i followed by a space
      i++;
    }
    i--;
    cout << i << " ";   // Output the current value of i followed by a space
    i--;
  }
  i--;
  cout << i << endl;    // Output the current value of i followed by a newline

  return 0;
}
```

What gets printed? _____

4. Why is the use of a traversal pointer to cycle through all the elements of a C/C++ multidimensional array almost always more efficient than using standard array indexing?

What question would you most like to see on the Midterm?