1 Pointers in C

A pointer is a typed variable that holds the memory address of a variable of the appropriate type. The unary operator & returns the memory address (location) of a variable. If $i$ is of type int, then $&i$ is of type pointer to int. The unary operator * (not to be confused with the binary multiplication operator) has two context-dependent meanings:

1. In a variable declaration, * indicates that the variable's type is pointer to some base type — see the example below.

2. In an expression, * “dereferences” a pointer variable, chasing the memory address to the variable to which the pointer points. Again, see the example below.

Note that if $i$ is type pointer to int then $*i$ is type int. Thus, the * and & operators are inverses of each other.

Example:

```c
double x = 0.0;
double *dblPtr; /* pointer to double */
int i = 1;
int *intPtr; /* pointer to int */
int **intPtrPtr; /* pointer to pointer to int --- intPtrPtr hold the memory address of another pointer */
dblPtr = &x;
intPtr = &i;
intPtrPtr = &intPtr;
```

Given the example code above, what is the value of each of the following expressions:

1. $i$, $x$.

2. Assign appropriate values to the following expressions so that you can assign values to the expressions in the following questions:

   (a) $&x$
   (b) $&i$
   (c) $&intPtr$
3. `dblPtr, *dblPtr`.
4. `intptr, *intptr`.
5. `intptrPtr, *intptrPtr, **intptrPtr`

## 2 Base & Offset Addressing

Consider the following C program (available on the class web site as `baseoffset.c` for copy & paste purposes):

```c
#include <stdio.h>

int main()
{
    int offset;
    int *base;
    int A[8] = { 0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC, 0xDE, 0xF0 };

    offset = 0;
    base = &A[0];

    printf("Legend:
 <Variable>: <Value> @ <Address>

");  
    printf("offset: %X @ %X\n", offset, &offset);
    printf("base: %X @ %X\n", base, &base);

    for (offset = 0; offset < 8; offset++)
        printf("A[%d]: %X @ %X\n", offset, *(base + offset), base + offset);

    return 0;
}
```

1. Note the use of base & offset addressing in the body of the for loop:

   ```c
   printf("A[%d]: %X @ %X\n", offset, *(base + offset), base + offset);
   ```

   (a) What is the type of `base`? What type of data does it hold?

   (b) What is the type of `offset`? What type of data does it hold?

   (c) What is the type of the expression `base + offset`?

   (d) What is the difference between the expression `*(base + offset)` and the expression `base + offset`?

2. Using NX, logon to phoenix.

3. Download and compile the program:

   ```bash
   % gcc -o baseOffset baseOffset.c
   ```

4. Run the program a couple times, noting any differences between the run outputs:
% ./baseOffset

5. Answer these questions:
   
   (a) How do the outputs differ, run-to-run?
   
   (b) All variables used by the program are word-sized (32 bits).
       
       Is memory word addressable or byte addressable?
       
       Are the variables word-aligned?

6. Interpret the output.

7. Draw a memory map showing how the variables are laid-out in memory and the relationships between the variables.