1. **True/False.** Circle the appropriate choice on this sheet (there are no trick questions).

   (a) T  In x86 assembly programming, the EBP register is commonly used to point into the stack frame of the currently executing procedure.

   (b) F  Global variables declared in C cannot be referenced within x86 assembly code.

   (c) T  In C, dereferencing a NULL pointer will cause a program to crash.

   (d) T  In x86 assembly, the `ret` instruction pops an address off the stack and performs a jump to that address.

   (e) F  Access to main memory (RAM) is generally faster than access to cache memory.

   (f) F  A caller-saved register must be saved before a procedure call, regardless of whether the register’s value is needed or not after the call.

   (g) T  In C, the call `malloc(n)` allocates a block of n bytes in memory and returns the address of the block.

   (h) F  In C, the declaration `int x[10];` does not actually allocate an array of ten integers.

   (i) T  In C, the line `#define FOO 17` does not cause any space to be allocated in memory for FOO.

   (j) F  In C, executing the statements `p = &x; p++;`, assuming both p and x have been declared correctly, will result in x being incremented by 1.

2. Write the x86 assembly code corresponding to the following C code.

```c
void foo(int x, int y, int *z)
{
    if (x==y)
        *z = *z + 1;
    else
        *z = *z - 1;
}
```
3. (a) In C, define a structure type `NODE` for a node in a binary tree, where each node has an integer `val` field in addition to two child fields.

**Answer:**

```c
typedef struct node {
    int val;
    struct node *left;
    struct node *right;
} NODE;
```

(b) Write the code for the recursive function `int max(NODE *t);`

that, given a pointer `t` to the root of a tree, returns the maximum value among the `val` fields of the nodes in the tree.

**Answer:** There are many ways to write this. Below is a reasonably compact version. Note that it avoids returning some “dummy” maximum value for an empty tree.

```c
int max(NODE *t)
{
    if (t == NULL) {
        printf("Error: NULL TREE\n");
        exit(1);
    }
    int val_left = (t->left) ? max(t->left) : t->val;
    int val_right = (t->right) ? max(t->right) : t->val;
    if (val_left > val_right)
        return (t->val > val_left) ? t->val : val_left;
    else
        return (t->val > val_right) ? t->val : val_right;
}
```
4. Recall that the x86 addressing modes are as follows:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Intel</th>
<th>AT&amp;T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute</td>
<td>MOV EAX, [0100]</td>
<td>movl 0x0100, %eax</td>
</tr>
<tr>
<td>Register</td>
<td>MOV EAX, [ESI]</td>
<td>movl (%esi), %eax</td>
</tr>
<tr>
<td>Reg + Off</td>
<td>MOV EAX, [%EBP-8]</td>
<td>movl -8(%ebp), %eax</td>
</tr>
<tr>
<td>R*W + Off</td>
<td>MOV EAX, [%EBX+4 + 0100]</td>
<td>movl 0x100(%ebx,4), %eax</td>
</tr>
<tr>
<td>B + R*W + O</td>
<td>MOV EAX, [%EDX + %EBX*4 + 8]</td>
<td>movl 0x8(%edx,%ebx,4), %eax</td>
</tr>
</tbody>
</table>


(a) Suppose `arr` is an array declared as a local variable in a procedure and starts at EBP-100 on the stack. Suppose `index` is an integer that resides in the ECX register. Write a single x86 assembly instruction to put the value of `arr[index]` into the EAX register.

Answer:

#AT&T Syntax
mov -100(%ebp,%ecx,4),%eax

#Intel Syntax
mov eax,[ebp+ecx*4-100]

(b) Suppose `p` is a pointer to a `PERSON` structure, defined in C by:

```c
typedef struct person { char *name; struct person *next; int age; } PERSON;
```

Assuming the value of `p` is in EDX, write two x86 assembly statements that, together, cause the value of `p->next->age` to be put in the EAX register.

Answer:

#AT&T Syntax
mov 4(%edx),%eax
mov 8(%eax),%eax

#Intel Syntax
mov eax,[edx+4]
mov eax,[eax+8]

(c) Suppose that `arr`, `index`, and `p` are exactly as described in the previous two questions.

i. Can the value of `arr[index]` be copied to `p->age` in a single x86 instruction? If so, give the instruction. If not, state why not. [Please note that this question refers to `p->age`, unlike the previous question.]

Answer: No. A single x86 instruction cannot take two memory locations as operands.

ii. If your answer to the previous question was “no”, then write the fewest x86 instructions that, together, copy the value of `arr[index]` into `p->age`.

Answer:

#AT&T Syntax
mov -100(%ebp,%ecx,4),%eax
mov %eax,8(%edx)

#Intel Syntax
mov eax,[ebp+ecx*4-100]
mov [edx+8],eax
5. Write a C procedure that corresponds to the following x86 assembly code. Be sure your C code reflects the parameter(s) and local variable(s) used in the assembly.

```c
int f(int x)
{
    int y = 6;  // y is on the stack at ebp-4
    int z = x;  // z is in ecx
    while (y > 0) {
        z = y * z;
        y--;
    }
    return z;
}
```