Test I Review

• Topic: C programming language
• Lecture Notes
• Assignment 1 & 2

Bitwise Operators

• int x = 1;
• int y = 2;
• x && y // 1
• x & y // 0
• x || y // 1
• x | y // 3
• x ^ y // 3
• x >> 1 // 0

Increment and Decrement

• Pre-increment and decrement operators:
  — Increase/decrease the operand before it is used.
  — ++x is equivalent to x = x + 1 (same applies to - -x)
  — int count = 0; printf("%d", ++count);
    // add 1 to count, print 1
• Post-increment and decrement operators.
  — Increase/decrease the operand after it is used.
  — x++ is equivalent to x = x + 1 (same applies to x--)
  — int count = 0; printf("%d", count++);
    // print 0, add 1 to count
Micro vs. inline vs. regular functions

<table>
<thead>
<tr>
<th></th>
<th>macro</th>
<th>Inline function</th>
<th>Regular function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handled by</td>
<td>cpp</td>
<td>compiler</td>
<td>compiler + linker</td>
</tr>
<tr>
<td>Code error checking?</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Run-time efficiency</td>
<td>best</td>
<td></td>
<td>worst</td>
</tr>
<tr>
<td>Compile-time efficiency</td>
<td>worst</td>
<td></td>
<td>best</td>
</tr>
<tr>
<td>Executable code size</td>
<td>largest</td>
<td></td>
<td>smallest</td>
</tr>
<tr>
<td>Placed in .h or .c</td>
<td>.h</td>
<td>.h</td>
<td>.h</td>
</tr>
</tbody>
</table>

Address Arithmetic

- If two pointer variables point to the elements of the same array, there are notions of subtraction and comparisons between the two pointers.
  
  \[ p - q : \text{the difference between array index that p and q point to.} \]

  \[ p < q : \text{true if p points to earlier elements of the array than q does} \]

Create 2-D matrix using Loops

```
const int row=3;
const int col=3;
int i, j;
int value=1;
int table[3][3];

for (i=0; i < row; i++)
  for (j=0; j < col; j++)
    table[i][j]=value++;
```

Switch Statement

```
switch (c) {
  case '0': case '1': case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '9':
    digit[c-'0']++; break;
  case ' ': case '
    nwhite++; break;
  default:
    nother++; break;
}
```

The statement is executed when one of the 10 cases is matched.

Without "break", all statements followed (nwhite++; break;) in the switch statement will be executed.
Loop Structures

<table>
<thead>
<tr>
<th>Loop</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i=5; int sum=0; while(i &lt;=10); { sum=sum+i; i++; } printf(&quot;%d %d \n&quot;,sum,i);</td>
<td>Print nothing. The code runs into infinite loop as i never gets updated.</td>
</tr>
<tr>
<td>int i=5; while (i-- &gt; 0); printf(&quot;%d \n&quot;,i);</td>
<td>1</td>
</tr>
<tr>
<td>int i; for(i=5; i!=0; i=i-2) printf(&quot;%d \n&quot;,i);</td>
<td>5 3 1 -1 infinite loop</td>
</tr>
</tbody>
</table>

Passing Structure as Function Parameter

- Structure parameters are passed by values:
  - The entire struct argument is copied over to the struct parameter.
  - The changes made in the struct argument are not visible outside the function.

```c
struct point { int x; int y; };
struct point a = {10,10}; struct point b={20,20};
addpoints(a,b) //a.x? a.y?
```

//addpoints: add two points
struct point addpoints(struct point p1, struct point p2) {
  p1.x += p2.x; p1.y += p2.y; return p1;
}

Array Name as Function Argument

- Passing array names as function argument is equivalent to passing pointer variables to the function.

```c
fun1(int a[]) main() {
  //a is a pointer variable
  *(a++)=5; //a points a[1]
  a[0]=6; //a[1]
  (*a)++; //a[1]++
}
fun2(int* a) {
  //a is a pointer variable
  *a=8;
  *(a+1)=9;
  a[1]++;
}
```

Function Returning Array

- Functions cannot return local arrays variables, which have local scope.
- The following won’t work!!

```c
char *itoa(int n) {
  char rebuf[25]; //char *rebuf = malloc(25);
  if(rebuf == NULL) return NULL;
  sprintf(rebuf, "%d", n);
  return rebuf;
}
```
Function Returns Structures

- Functions can return local `struct` variable, which can be assigned to a variable:
  ```
  struct point a = makepoint(10,10);
  ```
- This is because a new copy of the local struct variables is returned from the function.

```
struct point
  makepoint(int x, int y)
  {
    struct point temp;
    temp.x = x;
    temp.y = y;
    return temp;
  }
```