1 C Introduction

C is syntactically very similar to Java, but there are a few key differences of which to be wary:

- C is function oriented, not object oriented, so there are no objects.
- C does not automatically handle memory for you.
  - In the case of stack memory (things allocated in the “usual” way), a datum is garbage immediately after the function in which it was defined returns.
  - In the case of heap memory (things allocated with malloc and friends), data is freed only when the programmer explicitly frees it.
  - In any case, allocated memory always holds garbage until it is initialized.
- C uses pointers explicitly. \*p tells us to use the value that p points to, rather than the value of \( p \), and \&x gives the address of \( x \) rather than the value of \( x \). See the following example (the following addresses were chosen arbitrarily). On the left we see a diagram of pointers and memory that may help you visualize pointers. On the right, we see how those “boxes and arrows” are really represented.

<table>
<thead>
<tr>
<th>0xFFFFFFFF</th>
<th>0xFFFFFFFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>0xFFFFFFFF</td>
</tr>
<tr>
<td>0x0F320900</td>
<td>0x0F320900</td>
</tr>
<tr>
<td>0x0F320904</td>
<td>0x0F320904</td>
</tr>
<tr>
<td>0x0F3209AC</td>
<td>0x0F3209AC</td>
</tr>
<tr>
<td>0x0F32090B</td>
<td>0x0F32090B</td>
</tr>
<tr>
<td>0x61C</td>
<td>0x61C</td>
</tr>
<tr>
<td>0x2A</td>
<td>0x2A</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

Let’s assume that int* p is located at 0xF320904 and int x is located at 0xF32090B. As we can observe:

- *p should return 0x2A (42\textsubscript{10}).
- p should return 0xF3209AC.
- x should return 0x61C.
- \&x should return 0xF3209B0.

Let’s say we have an int **pp that is located at 0xF320900. What would pp return? How about *pp? What about **pp?

There are other differences in C of which you should be aware of, but this should be enough for you to get your feet wet.

2 Uncommented Code? Yuck!

The following functions work correctly (note: this does not mean intelligently), but have no comments. Document the code to prevent it from causing further confusion.
1. /*
   *
   */
   int foo(int *arr, size_t n) {
       return n ? arr[0] + foo(arr + 1, n - 1) : 0;
   }

2. /*
   *
   */
   int bar(int *arr, size_t n) {
       int sum = 0, i;

       for (i = n; i > 0; i--) {
           sum += !arr[i - 1];
       }

       return ~sum + 1;
   }

3. /*
   *
   */
   void baz(int x, int y) {
       x = x ^ y;
       y = x ^ y;
       x = x ^ y;
   }

3 Programming with Pointers

Implement the following functions so that they perform as described in the comments.

1. /* Swaps the value of two ints outside of this function. */

2. /* Increments the value of an int outside of this function by one. */

3. /* Returns the number of bytes in a string. Does not use strlen. */
4 Problem?

The following code segments may contain logic and syntax errors. Find and correct them.

1. /* Returns the sum of all the elements in SUMMANDS. */
   int sum(int* summands) {
     int sum = 0;
     for (int i = 0; i < sizeof(summands); i++)
       sum += *(summands + i);
     return sum;
   }

2. /* Increments all the letters in the string STRING, held in an array of length N.
   * Does not modify any other memory which has been previously allocated. */
   void increment(char* string, int n) {
     for (int i = 0; i < n; i++)
       *(string + i)++;
   }

3. /* Copies the string SRC to DST. */
   void copy(char* src, char* dst) {
     while (*dst++ = *src++);
   }

4. /* Overwrites an inputted string with ‘61C is awesome!’ if there’s room.
   * Does nothing if there is not. Assume that srcLength correctly represents
   * the length of src. */
   void CS61C(char* src, size_t srcLength) {
     char *srcptr, replaceptr;
     char replacement[15] = ‘61C is awesome!’;
     srcptr = src;
     replaceptr = replacement;
     if (srcLength >= 15) {
       for (int i = 0; i < 15; i++)
         *srcptr++ = *replaceptr++;
     }
   }