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.\texttt{*p} tells us to use the value that \texttt{p} points to, rather than the value of \texttt{p}, and \texttt{&x} gives the address of \texttt{x} rather than the value of \texttt{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>Address</th>
<th>Value</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFFFF</td>
<td>...</td>
<td>0xFFFFFFFF</td>
<td>...</td>
</tr>
<tr>
<td>0xF93209B0</td>
<td>\texttt{x=0x61C}</td>
<td>0xF93209B0</td>
<td>0x61C</td>
</tr>
<tr>
<td>0xF93209AC</td>
<td>0x2A</td>
<td>0xF93209AC</td>
<td>0x2A</td>
</tr>
<tr>
<td>0xF9320904</td>
<td>\texttt{P}</td>
<td>0xF9320904</td>
<td>0xF93209AC</td>
</tr>
<tr>
<td>0xF9320900</td>
<td>\texttt{PP}</td>
<td>0xF9320900</td>
<td>0xF9320904</td>
</tr>
<tr>
<td>0x00000000</td>
<td>...</td>
<td>0x00000000</td>
<td>...</td>
</tr>
</tbody>
</table>

Let’s assume that int* \texttt{p} is located at 0xF9320904 and int \texttt{x} is located at 0xF93209B0. As we can observe:

- \texttt{*p} should return 0x2A (42\textsubscript{10}).
- \texttt{p} should return 0xF93209AC.
- \texttt{x} should return 0x61C.
- \texttt{&x} should return 0xF93209B0.

Let’s say we have an int **\texttt{pp} that is located at 0xF9320900. What would \texttt{pp} return? How about \texttt{*pp}? What about \texttt{**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. /* Returns the sum of the first N elements in ARR. */
   int foo(int *arr, size_t n) {
     return n ? arr[0] + foo(arr + 1, n - 1) : 0;
   }
2. /* Returns -1 times the number of zeroes in the first N elements of ARR. */
   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. /* Does nothing. */
   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. */

   void swap(int *x, int *y) {
      int temp = *x;
      *x = *y;
      *y = temp;
   }

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

   void plus_plus(int *x) {
      (*x)++; // or: x[0]++;
   }

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

   int mystrlen(char* str) {
      int count = 0;
      while(*str++) {
         count++;
      }
      return count;
   }
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(int* summands, unsigned int n) {
     int sum = 0;
     for (int i = 0; i < sizeof(summands); i++) // for (int i = 0; i < n; 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++) // for (i = 0; string[i] != 0; i++)
       *(string + i)++; // string[i]++; or (*(string + i))++;
     // consider the corner case of incrementing 0xFF
   }

3. /* Copies the string SRC to DST. */
   void copy(char* src, char* dst) {
     while (*dst++ = *src++);
   }
   // This code has no errors.

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 *srcptr, *replaceptr;
     char replacement[16] = ’61C is awesome!’;
     srcptr = src;
     replaceptr = replacement;
     if (srcLength >= 16) {
       for (int i = 0; i < 16; i++)
         *srcptr++ = *replaceptr++;
     }
   }
   // ‘char *srcptr, replaceptr’ initializes a char pointer and a char. Not two char pointers.
   // ‘char *srcptr, replaceptr’ is not the same as ‘char *srcptr, *replaceptr’.