Pointers

What are pointers?

- A mechanism to keep the address
- Pointer type variables hold nothing but an address in memory where the actual contents of the variable are to be found
- One of the most complex and most powerful paradigms in a programming language

Pointer variable declarations and initialization

- Indirection
  - Indirect reference to a value contained in a variable
- Pointer variables must be declared before use
- Definition syntax

```c
int *count_ptr, count;
```

- The above definition defines two variables – `count_ptr` which is a pointer to a variable of type `int`, and `count` which can hold an integer value
- If more than one variable is to be defined as a pointer, each such variable should be prefixed with a `*` in the definition
- It is a good idea to use `_ptr` as a suffix to indicate that a variable is a pointer type
- Pointers can be initialized when they are defined or in an assignment statement
  - The only initialization values permissible for pointers are 0, NULL, or an address
  - NULL is a symbolic constant defined to be 0 in `stdio.h`
  - Assigning an arbitrary integer to a pointer is not permissible (leads to a run-time error)

Pointer operators

- The address operator (`&`)
  - Has been used in the `scanf` statement
  - Using the above definition of `count` and `count_ptr`, the following assigns the address of `count` to `count_ptr`

```c
count_ptr = &count;
```

  - This operator cannot be applied to constants, expressions, variables defined with the storage class `register`
- The dereferencing operator (`*`)
  - Also known as the indirection operator
  - Returns the value contained in the storage location pointed to by its operand
  - After executing the last assignment statement, both `count` and `*count_ptr` return the contents of `count`
  - The operation itself is known as dereferencing a pointer
– This operation is almost the biggest source of run-time errors in C

• Both * and & operators are complements of one another

\[ *x \equiv &*x \]

Calling functions by reference

• We have already noted that all parameters in C are passed to the functions using call by value
• Also functions are constrained by the fact that only one value can be returned to the caller
• Pointers and indirection operators provide a work around this limitation of C
• Let us write a function to exchange the value in two variables

```c
/*** xch.c : Exchange two integer variables */
void exchange ( int * x, int * y )
{
    int tmp; /* Temporary variable */
    tmp = *x;
    *x = *y;
    *y = tmp;
}
```

• The function is called by a statement

```c
    exchange ( &x, &y );
```

• If you are passing arrays, you do not need to prefix & to the name of the array because the name itself is a pointer
• Also look at the examples in the book (Figs. 7.6 and 7.7; p. 265)
• The prototype for variables passed by reference should contain an asterix after the type of variable; The prototype for exchange function is

```c
void exchange ( int *, int * );
```

Using the const qualifier with pointers

• Use of pointers forces the parameters to be passed by reference
• May cause problems if the function accidently modifies a variable that was not intended to be modified
• Such behavior could be avoided by the use of the const qualifier
• As an example, consider a function

```c
void print_array ( int a[], int n )
```
• Since this function does not need to modify the elements in the array, you are better off passing both the parameters as const
  – It may not really be necessary to qualify \( n \) with \texttt{const} because it is automatically passed by value
• You should also check the function prototype to see if the values passed to the function get modified
• \texttt{const} provides the efficiency of call-by-reference (no overhead in copying the parameters) and the protection of call-by-value (no modification of data allowed)

• Non-constant pointer to constant data
  – A pointer that can be modified to point to any data item of appropriate type
  – Does not allow the modification of elements pointed to by the pointer
  – Exemplified by the function \texttt{print\_array} which should be declared as
    \begin{verbatim}
    void print_array ( const int * a, int n )
    \end{verbatim}
  – The declaration using asterix is the same as the one using square brackets because of equivalence between pointers and arrays
  – The function itself could be written as
    \begin{verbatim}
    void print_array ( const int * a, int n )
    {
        int counter = 0;
        do
            printf ( "%10d\n", *a++ );
        while ( ++counter < n );
    }
    \end{verbatim}
  – Notice how each element is accessed in the array by using the pointer notation; also, that the pointer variable itself is incremented to point to the next address in the array

• Constant pointer to non-constant data
  – Pointer itself always points to the same location and cannot be modified
  – However, the data itself can be modified (even when accessed through the pointer)
  – The declaration will be given by
    \begin{verbatim}
    void foo ( int * const x )
    \end{verbatim}

• Constant pointer to constant data
  – The most restrictive form of parameter passing
  – Cannot modify pointer or data
  – The declaration will be given by
    \begin{verbatim}
    void foo ( const int * const x )
    \end{verbatim}

\textbf{Bubble sort using call by reference}

• Reading assignment

\textbf{Pointer expressions and pointer arithmetic}
• The `sizeof()` operator
  – Can be used to determine the number of bytes in any data type
  – The data type can be simple (such as `int`) or complex (such as `int[100]`)

• Pointers allow limited arithmetic operations
  – You can use `++` or `--` operators to increment or decrement a pointer
  – You can use `+=` or `-=` to add or subtract an integer from a pointer
  – Finally, you can use `+` or `-` to add or subtract one pointer from another
  – For each of these operations, pointers add or subtract the number of bytes depending upon the size of the data type

**Relationship between pointers and arrays**

• C implements arrays using pointers and the fact can be exploited in the programs