Data Structures and Program Design in C

Topic 5: More on Pointers, Structs, Unions and Macros

Dr Manjusri Wickramasinghe

Outline

- Types of Pointers
- Strings
- Structures and Unions
- Macros
- Static keyword

Types of Pointers

- Pointers are classified into many types based on how they are defined and what they point to. A few notable names are as follows:
    - Void pointers
    - Constant pointers
    - Pointers to constants
    - Array pointers
    - Function pointers
    - Null pointers
    - Wild pointers
    - Pointer to Pointer (Double Pointer)

Void pointers …(1)

- Void pointers are pointers that have no data type associated with them.

- Void pointer can hold an address of any type and can be typecast to any type.

```
void *<name>;
```

Void pointers …(2)

- Void cannot be dereferenced.

- Pointer arithmetic
  - C standard does not allow for void pointer arithmetic
  - GNU C does this by considering the size of the void pointer as one (1).

- To type case a void pointer the following syntax is used.

```
(<data_type>*)<void_pointer_name>;
```

Void pointers …(3)

- Void pointers enable the implementation of generic functions.
- Mainly used in implementing data structures.
- Discarding type safety is considered a disadvantage of void pointers.
  - Requires manual type checking and casting.

Function Pointers …(1)

- Functions pointers are pointers to functions declared using the following syntax.

```
return_type (*ptr_name)(param 1, param 2, …);
```

- Function pointers point to the start of the executable code.

- Enables the calling of the function using the function pointer.

Function Pointers …(2)

- Enables the creation of generic functions that can be used with any functions provided it has the relevant method signature.

- Useful in implementing callback functions.
  - A reference of a function passed as a parameter to another function so it will be called through the other function.
  - Can be used to create wrappers around existing functions.

- Function pointers can create hard-to-read and complicated declarations.

Function Pointers …(3)

- To read complex declarations, the **spiral rule** is applied **clockwise**, starting from the unknown element.
  - When encountering the following replace it with the corresponding English statement
    1. [x] or [] → Array of X size or Array of undefined size
    2. (a, b) → function passing type a and type b returning
    3. * → pointer to
  - Repeat the above steps until all tokens are covered.
  - Resolve the parenthesis first.

Constants and Pointers …(1)

- There are three (03) variants of pointers with the constant keyword:
  - Constant pointers
  - Pointer to constant
  - Constant pointer to constant

Constants and Pointers …(2)

- Pointer to a constant is defined as follows:

```
const data_type *<pointer_name>;
```

- The data pointed by the pointer can only be read (read-only location) but cannot be changed using the pointer.

- The pointer, though read-only, can be reassigned to point to another memory location.

Constants and Pointers …(3)

- Constant pointers are defined as follows:

```
data_type *const <pointer_name>;
```

- Constant pointers will always point to the same memory location and, once assigned, cannot be changed.

- The constant pointers have to be assigned at initialization.

- The value at the memory location can be changed.

Constants and Pointers …(4)

- Putting both pointer to constant and constant pointers together will create a constant pointer to constant. The syntax is as follows:

```
const <data_type>* const <pointer_name>;
```

- Such pointers cannot be pointed to anywhere, and the data cannot be changed once assigned.

- Has to be initialized at the declaration.

Pointers and Arrays

- All elements in an array can be accessed using pointers and pointer arithmetic.

- Pointer array is where a homogenous collection of pointer variables are indexed as an array. The syntax is as follows:

```
pointer_type* array_name[array_size];
```

Null Pointers …(1)

- A null pointer is a pointer that does not point to any location.
- Null pointers can be obtained by assigning a pointer to a **zero (0)** or **NULL**.
- Printing a null pointer will result in different behaviours across different machines.

Null Pointers …(2)

- The main difference between null pointers and void pointers is that void pointers:
  - Points to a memory location
  - Not all void pointers are the same
  - Void pointer is a type

- Null pointers are useful to
  - Initialize pointer variables that have not been given an initial address.
  - To check for null before accessing a memory location.
  - To pass to functions when there is a not-memory address to pass as a parameter.

Wild Pointers

- Wild pointers are uninitialized and may point to an arbitrary memory location.

- Such pointers can make the program work unpredictably or cause crashes.

Strings … (1)

- C programming language does not have an inbuilt string data type.
- In a C programming language, the string is a sequence of characters terminated with a null (‘\0’) character.
- Declaring a string is equivalent to declaring a one-dimensional array of characters of a specified size.

```
char <string_name>[array_size]
```

Strings … (2)

- If a string of size n needs to be stored, the array size should be n+1 accounting for the null termination character.

- Examples
  - `char str1[] = "SCS 1301 – DSA in C";`
  - `char str2[25] = "SCS 1301 – DSA in C";`
  - `char str3[] = {'T', 'E', 'S', 'T', '\0'};`

- When assigning character by character, as in 'str3', the **null character has to be inserted at the end**.

- For the double quotation mark initialization as in 'str1' and 'str2', the null character is assumed.

Strings …(3)

- Strings can be formatted in various formats when producing output using `printf(…)` and parsing them into a program using `scanf(…)`.

- ‘%s’ formatting directive is used to work with strings.

- Scansets are used to process only those characters specified in the set when parsing a string using `scanf(%[…]s, string)`.

Strings …(4)

- Examples
  - `%[A-Z]s` → Only will store the capital letters.
  - `%[^a]s` → Stores characters until it finds the first ‘a’ and stops. ‘a’ is not stored.
  - `%[A-Z_a,b,c]s` → Only will store capital letters, underscore, ‘a’, ‘b’, and ‘c’.
  - `%[0-9]s` → Only stores number characters.

- Strings works with a number of inbuild string manipulation functions defined in <string.h>

Structures …(1)

- Structures are a user defined type in the C programming language.

- Syntax

```
struct <name>
{
        <data_type 1> <variable_name 1>;
        <data_type 2> <variable_name 2>;
        …
} [variable name 1, variable name 2 …];
```

Structures …(2)

- Logically groups related heterogenous items under a single unit, where all items can be separately accessed.

- All items under a structure are called ‘members’ of that structure.

- To declare a variable for a given struct the following syntax is used.

```
struct <struct_name> <variable_name>;
```

- To access a member:

```
<varaiable_name>.<member_name>;
```

Structures …(3)

- To define a pointer to a structure:

```
struct <struct_name> *ptr;
```

- To assign to a pointer to a structure:

```
struct <struct_name> temp;
ptr = &temp;
```

- To access an element using pointers:

```
ptr-> <member_name>
```

Structures …(4)

- Structure allocates memory for each data member.

- However, when the **`sizeof(…)`** operator is used on a structure, the size in memory can be different from the total size of data members due data alignment of the computer.

- The members of structures are allocated within a contiguous block of memory.

Unions …(1)

- Union is a user defined data type that allows several types of data types to be stored in the same memory location.

- Since all members are stored in the same memory location, a **single member** can store data at a give instance.

- Syntax of a union is as given below:

```
union <union_name>
{
        <data_type> <member_name1>;
        <data_type> <member_name1>;
        …
} [variable_name];
```

Unions …(2)

- All members cannot be initialized at the same time. Since all members point to the same memory location, change in one would be reflected on the other.
- Size of the union is equal to the largest data member in the union.
- Primary application of the union is to save memory.
- Uses similar syntax to structures when used as a pointers.

Macros …(1)

- Macros are preprocessor directives defined by the #define directive.
- Macro is a name given to a piece of code such that when the preprocessor encounters it in the program, it will be replaced by the code defined.
- Macros are not terminated by a semicolon.
- The main use case of a macro is to increase program efficiency by defining a constant and reusing it multiple time rather than writing the same piece of code repeatedly.

Macros …(2)

- Two (2) types of macros
  - Object like macros
  - Function like macros

- Object like macros have a defined value and is considered a constant numerical value.

- Function is defined in function like macros and arguments are passed by the #define directive.

Macros …(3)

- There are predefined macros in C such as `__DATE__`, `__TIME__`, etc.

- Examples
  - `#define add(a, b) (a + b)`
  - `#define min(a, b)(((a) < (b))?(a):(b))`
  - `#define PI 3.146`
  - `#define area(r) (PI*(r)*(r))`

`static` keyword ...(1)

- Static keyword can be applied to both variables and functions.
- When applied to variables it has the property of preserving its value even after they are no longer in scope.
- Syntax for static variables:

```
static <data_type> <var_name> = value;
```

`static` keyword ...(2)

- Properties
  - Static variable remains in memory as long as the program is running.
  - Static variables are placed in data memory as opposed to stack memory.
  - Static variables are initialized to zero in the case they are not initialized.
  - Static variables should not be declared inside a structure. However, a static structure is acceptable.

`static` keyword …(3)

- Static functions can be declared using the following syntax.

```
static <return_type> <function_name>([params] …)
{
        …
}
```

- Static functions are restricted to files they are declared in and mainly used when the same function name is to be used in other files.

Questions?