

CSCI 2410 Systems Programming

Fall 2026

Instructor: Jie Zhou

Department of Computer Science
George Washington University



Slides materials are mostly credited to Gabriel Parmer and Sibin Mohan at GWU.

Pointer Cast

- Can cast from one pointer type to another
- Between any two pointers
 - But cast between different pointer types needs to be *explicit*.
 - *Otherwise there will be compiler warnings.*
- A pointer itself is always the same size, e.g., 8 bytes on a 64-bit system
- Does not change the memory underneath
 - but, changes pointer arithmetic.

Cast between Regular Pointers and void *

- Can cast any pointer to void *
- Can also cast void * to any pointer
- Cannot dereference a void *

```
void *pv = malloc(32);  
int *pi = (int*) pv; // the (int *) before pv is optional  
double *pd = (double*) pv; // (double *) before pv is also optional.  
char *pc = pi; // (char *) before pi is needed; there would be a warning otherwise.
```

Pointer Arithmetic

- Consider an integer array, `int a[5];`
- 5 ints, each of 4 bytes
- `a[1]` (or `*(a + 1)`) is:



Function Pointers

Function

- Functions are also a kind of data, and they have *types* too.

```
void foo(int i, double d){...}
```

- Type of foo: a function that
 - takes one `int` argument and one `double` argument
 - returns nothing
- Most times, we call a function by its name.
- But sometimes, we need to decide *which* function to call at execution time.

Example: Bubble Sort

```
// Sorting ints
void bubble_sort_int( int array[], unsigned int array_size )
{
    for( unsigned int i = 0 ; i < array_size-1 ; ++i )
    {
        for( unsigned int j = 0 ; j < array_size-i-1 ; ++j )
        {
            if( array[j] > array[j+1] )
            {
                int temp = array[j] ;
                array[j] = array[j+1] ;
                array[j+1] = temp ;
            }
        }
    }
}
```

Example: Bubble Sort

- The bubble sort works for arrays of ints. But what if we want to sort an array of doubles?

```
// Sorting ints
void bubble_sort_int( double array[], unsigned int array_size )
{
    for( unsigned int i = 0 ; i < array_size-1 ; ++i )
    {
        for( unsigned int j = 0 ; j < array_size-i-1 ; ++j )
        {
            if( array[j] > array[j+1] )
            {
                double temp = array[j] ;
                array[j] = array[j+1] ;
                array[j+1] = temp ;
            }
        }
    }
}
```

Example: Bubble Sort

- But what if we want to sort an array of characters? Strings? floats? even customized struct? Do we write one function for each?

```
void bubble_sort_char( char array[], int array_size ){...}  
void bubble_sort_strings( char* array[], int array_size ){...}  
void bubble_sort_float( float array[], int array_size ){...}  
void bubble_sort_struct_student( struct student array[], int array_size ){...}
```

- Tedious work. And difficult to maintain.
- What if we don't know which one will be needed until run time?
 - e.g., users may input either an array of `int` or an array of `double`.

Two Improvements for Many Versions of Bubble Sort

Suppose we want a program that may run bubble sort for an array of data.
We want:

- Share as much of the core code as possible
- Support deciding which function to call at execution time

Generic Bubble Sort

- A sorting algorithm needs two key components: (1) compare and (2) swap

```
if( is_greater(a, b) ) // If a is larger than b
    swap(a, b)         // swap their values
```

- Would not it be great if we can could something like this:

```
// Sorting ints | Generic
void generic_bubble_sort( array, ...? ) {
    for( unsigned int i = 0 ; i < array_size-1 ; ++i ) {
        for( unsigned int j = 0 ; j < array_size-i-1 ; ++j ) {
            if( is_greater( array[j], array[j+1] )
                swap(array[j], array[j+1])
        }
    }
}
```

Generic Bubble Sort

```
// Sorting ints | Generic
void generic_bubble_sort(array, ...?) {
    for( unsigned int i = 0 ; i < array_size-1 ; ++i ) {
        for( unsigned int j = 0 ; j < array_size-i-1 ; ++j ) {
            if( is_greater( array[j], array[j+1] )
                swap(array[j], array[j+1])
        }
    }
}
```

- What is the type of the array?
- We want to make the array *generic*, so we define it as a `void *`.

```
void generic_bubble_sort(void *array, ...)
```

- We also need to know the size of the array.

```
void generic_bubble_sort(void *array, int array_size)
```

Generic Bubble Sort

```
// Sorting ints | Generic
void generic_bubble_sort(void *array, int array_size) {
    for( unsigned int i = 0 ; i < array_size-1 ; ++i ) {
        for( unsigned int j = 0 ; j < array_size-i-1 ; ++j ) {
            if( is_greater( array[j], array[j+1] )
                swap(array[j], array[j+1])
        }
    }
}
```

- We cannot do `array[j]`, because `void *` cannot be dereferenced.
- Standard C does not allow pointer arithmetic on `void *`.
 - Compilers may opt to support it, e.g., treating `void` as a `char`.
- Convert `void *` to a byte pointer before using it

```
unsigned char* base = array;
```

Generic Bubble Sort

```
// Sorting ints | Generic
void generic_bubble_sort(void *array, int array_size) {
    for( unsigned int i = 0 ; i < array_size-1 ; ++i ) {
        for( unsigned int j = 0 ; j < array_size-i-1 ; ++j ) {
            if( is_greater( array[j], array[j+1] )
                swap(array[j], array[j+1])
        }
    }
}
```

```
unsigned char* base = array ;
```

- `base + i` advances base by `i` bytes, not by `i` elements.
- So we need an `element_size` argument

Generic Bubble Sort

```
// Sorting ints | Generic
void generic_bubble_sort(void *array, int array_size, int element_size) {
    unsigned char* base = array ;

    for( unsigned int i = 0 ; i < array_size-1 ; ++i ) {
        for( unsigned int j = 0 ; j < array_size-i-1 ; ++j ) {
            if( is_greater(base + (j*element_size), base + ((j+1)*element_size)))
                swap(base + (j*element_size), base + ((j+1)*element_size))
        }
    }
}
```

What is still missing?

- `is_greater()`?
- `swap()`?
- We need to somehow provide the actual type for these functions to work on.
 - e.g., comparing `int` or `double`

Generic Bubble Sort

```
// Sorting ints | Generic
void generic_bubble_sort(void *array, int array_size, int element_size) {
    unsigned char* base = array ;

    for( unsigned int i = 0 ; i < array_size-1 ; ++i ) {
        for( unsigned int j = 0 ; j < array_size-i-1 ; ++j ) {
            if( is_greater(base + (j*element_size), base + ((j+1)*element_size)))
                swap(base + (j*element_size), base + ((j+1)*element_size))
        }
    }
}
```

Can `is_greater()` and `swap()` call different versions of compare and swap functions for different types of arrays?

- Which exact function to call depends on some function arguments.

Generic Bubble Sort

```
// Sorting ints | Generic
void generic_bubble_sort(void *array, int array_size, int element_size
                        <SOME_TYPE> is_greater, <SOME_TYPE> swap) {
    unsigned char* base = array ;

    for( unsigned int i = 0 ; i < array_size-1 ; ++i ) {
        for( unsigned int j = 0 ; j < array_size-i-1 ; ++j ) {
            if( is_greater(base + (j*element_size), base + ((j+1)*element_size)))
                swap(base + (j*element_size), base + ((j+1)*element_size))
        }
    }
}
```

- We can define the `is_greater` and `swap` arguments to be pointers to functions.
- We need `is_greater` to point to a function type that
 - takes two arguments to compare
 - return an `int` to indicate the comparison result

```
typedef int (*comparator_function_pointer)( void* l, void* r ) ;
```

Generic Bubble Sort

```
typedef int (*comparator_function_pointer)( void* l, void* r ) ;
```

- typedef associates a name with a type
- comparator_function_pointer refers to the function type
`int (*)(void *, void *)`
- Similarly, we can define swap:

```
typedef void (*swap_function_pointer)( void* l, void* r ) ;
```

Generic Bubble Sort

```
// Sorting ints | Generic
void generic_bubble_sort(void *array, int array_size, int element_size
                        <SOME_TYPE> is_greater, <SOME_TYPE> swap) {
    unsigned char* base = array ;

    for( unsigned int i = 0 ; i < array_size-1 ; ++i ) {
        for( unsigned int j = 0 ; j < array_size-i-1 ; ++j ) {
            if( is_greater(base + (j*element_size), base + ((j+1)*element_size)))
                swap(base + (j*element_size), base + ((j+1)*element_size))
        }
    }
}
```

Generic Bubble Sort

```
// Sorting ints | Generic
void generic_bubble_sort(void *array, int array_size, int element_size
                        comparator_function_pointer is_greater,
                        swap_function_pointer swap) {
    unsigned char* base = array ;

    for( unsigned int i = 0 ; i < array_size-1 ; ++i ) {
        for( unsigned int j = 0 ; j < array_size-i-1 ; ++j ) {
            if( is_greater(base + (j*element_size), base + ((j+1)*element_size)))
                swap(base + (j*element_size), base + ((j+1)*element_size))
        }
    }
}
```

- Function pointers are invoked exactly like regular functions.

Generic to Concrete Function

- We still need to implement concrete `is_greater()` and `swap()`.

```
int is_greater_than_int( void* l, void* r ) {  
    // cast it from void* to relevant type, int*  
    // since we cannot dereference void*  
    int* left = l ;  
    int* right = r ;  
  
    // compare and return result  
    if( *left > *right )  
        return 1 ;  
    else  
        return 0 ;  
}
```

Generic to Concrete Function

- We still need to implement concrete `is_greater()` and `swap()`

```
void swap_int( void* l, void* r ) {  
    // cast it from void* to relevant type, int*  
    // since we cannot dereference void*  
    int* left = l ;  
    int* right = r ;  
    int temp = *left ;  
  
    // swap  
    *left = *right ;  
    *right = temp ;  
}
```

Generic to Concrete Function

- If we want to compare double

```
int is_greater_than_double( void* l, void* r ) {  
    // cast it from void* to relevant type, int*  
    // since we cannot dereference void*  
    double* left = l ;  
    double* right = r ;  
  
    // compare and return result  
    if( *left > *right )  
        return 1 ;  
    else  
        return 0 ;  
}
```

Put Together

```
int main() {
    int my_array_int[] = { 2341, 8632, 3, 2344, 747645 } ;
    int array_size = 5 ;

    // calling the INTEGER version with the concrete integer comparator and swap
    generic_bubble_sort( my_array_int, array_size,
                        sizeof(int), /*element size*/
                        is_greater_than_int,
                        swap_int ) ;

    // calling the DOUBLE version with the concrete DOUBLE comparator and swap
    double my_double_array[] = {1.0, 9485.2, 34.567, 9383.243, 44.1 } ;
    generic_bubble_sort( my_double_array, array_size,
                        sizeof(double), /*element size*/
                        is_greater_than_double,
                        swap_double ) ;

    return 0 ;
}
```