Week 7

XMUT-NWEN 241 - 2024 T2

Systems Programming

Mohammad Nekooei

School of Engineering and Computer Science

Victoria University of Wellington

Content

- Function Returning a Pointer
- Pointer to Function

Function Returning a Pointer

Function Returning a Pointer

- Functions can return a pointer

- **Make sure that returned pointer points to a valid memory location**

```
float *find_max(float A[], int N)
{
   int i;
   float *the_max = &(A[0]);
   for (i = 1; i < N; i++)
      if (A[i] > *the_max) the_max = &(A[i]);
   return the_max;
}

int main(void)
{
   float scores[5] = {10.0, 8.0, 5.5, 2.0, 4.1};
   float *max_score;

   max_score = find_max(scores, 5);
   printf("%.1f\n",*max_score);
   return 0;
}
```

Pointer to function

Pointer to function

- Function code is stored in memory
  - Functions also occupy memory locations therefore every function has an address just like variables
  - Just like ordinary variables, the address of a function refers to its starting address

- C does not require that pointers only point to data, it is possible to have **pointers to functions**

Defining a function pointer

- Declaration:

```
return_type (*name)(param_types);
```

- Examples

```
int (*f)(int, float);
```

Pointer to a function that takes an `int` and `float` arguments, resp., and returns an `int`

```
int *(*f)(int, float);
```

Pointer to a function that takes an `int` and `float` arguments, resp., and returns a *pointer* to `int`

Using a function pointer

```
int F1(int i, float f)
{
        return i/f;
}

int main(void)
{
        /* f is a function pointer */
        int (*fp)(int, float);

        /* Assignment: let f point to F1 */
        fp = &F1; /* fp = F1; is also ok */

        /* Invocation */
        float a = fp(1, 2.0);
        /* This is equivalent to calling
        float b = F1(1, 2.0) */
        printf(“a = %f, b = %f”, a, b);
}
```

Comparing function pointers

- Can use the equality (==) operator
- Example:

```
/* f is a function pointer */
int (*fp)(int, float);

int F1(int i, float f)
{
        return i/f;
}

int main(void)
{
        /* Assignment: let f point to F1 */
        fp = &F1; /* fp = F1; is also ok */

        if(fp == &F1)
                printf("Points to F1\n");
}
```

Safety concerns

- What if uninitialized function pointer value is accessed
  - Safest outcome: memory error, and program is terminated
  - But what if the “garbage” value is a valid address?
    - Worst case: address contains program instruction –execution continues, with random results
    - Hard to trace the cause of the erroneous behavior

Usage of function pointers

- For implementing callback functions
  - Function pointer is passed as an argument to a function
  - The function will then invoke the passed function pointer at a given time

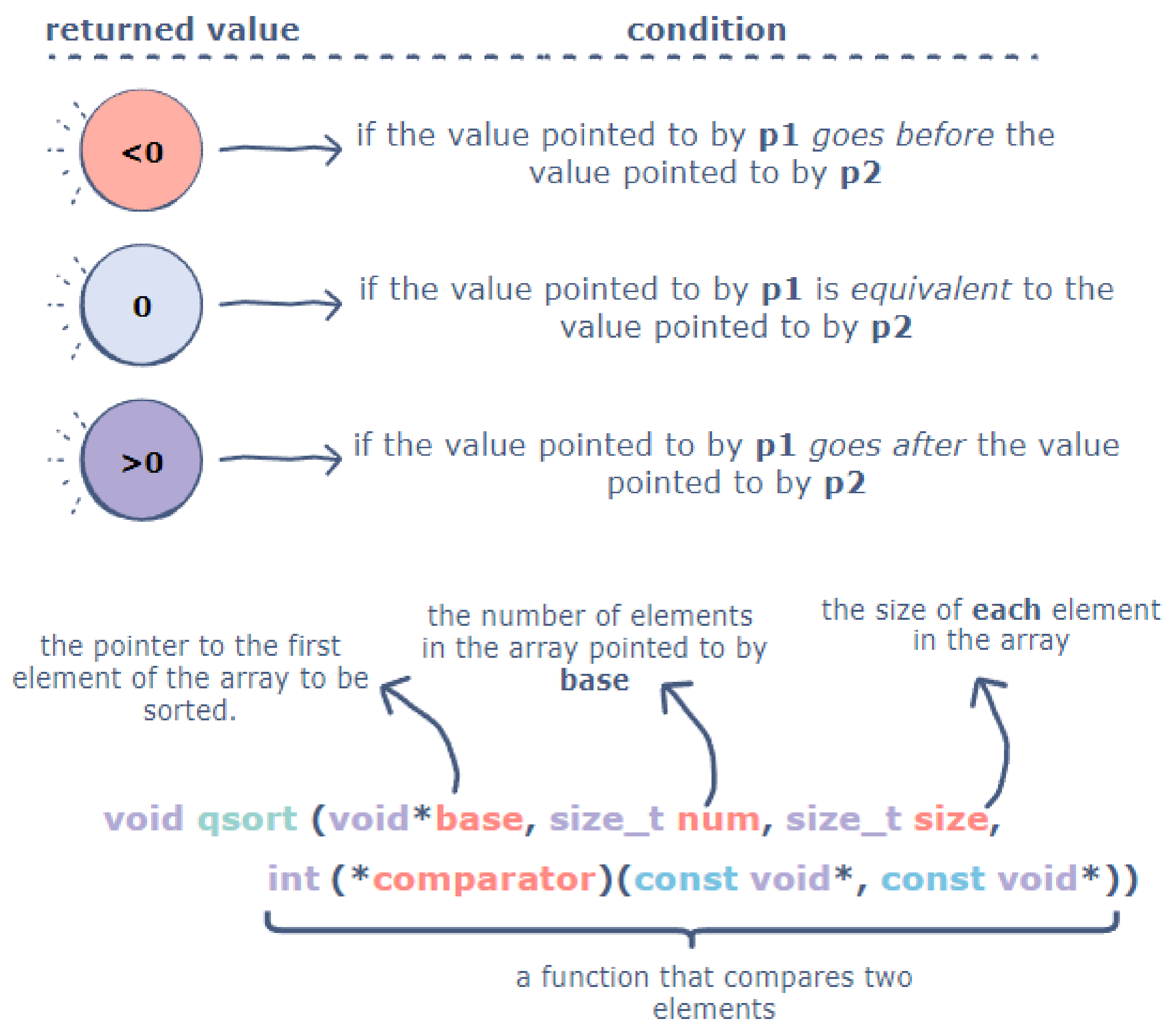
```
void qsort(void *base, size_t nitems, size_t size,
int (*compare)(const void *, const void*));
```

`base` – Pointer to the first element of the array to be sorted
`nitems` – Number of elements in the array pointed by base
`size` – Size in bytes of each element in the array.
`compare` – This is the function that compares two elements.

Function pointers: qsort example

```c
int arr[] = {20, 15, 36, -8, 2, 7};

int comparator (const void * p1, const void * p2)
{
  return (*((int*)p1) - *((int*)p2));
}
int main ()
{
  int size = sizeof(arr) / sizeof(arr[0]);
  printf("The unsorted array is: \n");
  for(int i = 0; i < size; i++)
  {
    printf("%d ", arr[i]);
  }
  qsort(arr, size, sizeof(int), comparator);
  printf("\nThe sorted array is: \n");
  for(int i = 0; i < size; i++)
  {
    printf("%d ", arr[i]);
  }
}
```



returned value	condition
<0	if the value pointed to by **p1** *goes before* the value pointed to by **p2**
0	if the value pointed to by **p1** is *equivalent* to the value pointed to by **p2**
>0	if the value pointed to by **p1** *goes after* the value pointed to by **p2**

```c
void qsort (void*base, size_t num, size_t size,
      int (*comparator)(const void*, const void*))
```

the pointer to the first element of the array to be sorted.
the number of elements in the array pointed to by **base**
the size of **each** element in the array
a function that compares two elements

Next lecture

- Structures