Function Pointers

Thumrongsak Kosiyatrakul
tkosiyat@cs.pitt.edu

Passing a Function as an Argument

- In C, a function can be passed as an argument to another function.
- This is useful when you want a function to call (use) a specific function.
- A function is just a sequence of machine instructions starting at a certain **address**
    - Therefore, we should be able to point to a function (pointer)
- However, functions have return value and argument
    - Pointer variable that can be used to point to a function needs return type and arguments

Function Pointer Declaration

- Recall pointer variable declarations:

```
int *x_ptr;
float *y_ptr;
char *c_ptr;
```

- Function pointer variable declarations needs return type and arguments
- Syntax:

```
returnType (* name)(arg1, arg2, ...)
```

  - `returnType` can be `void`, any types, or even pointer to a type
  - `name` can be any valid name in C
  - `(arg1, arg2, ...)` can be `void` or any types

Example

- A function pointer variable named `f_ptr` that can be used to point to a function that receives two integers as arguments and return an integer

```
int (* f_ptr)(int, int);
```

- A function pointer variable named `test` that can be used to point to a function that receive two arguments where the first argument is a pointer to integer and the second argument is an integer and return nothing can be declared as follows:

```
void (* test)(int *, int);
```

- A function named `foo` that return an integer and takes two arguments where the first argument is an integer and the second argument is **a function that takes an argument of type float and returns type integer** has the following signature:

```
int foo(int x, int (*bar)(float));
```

Case Study

- Suppose we want a function that return the maximum value from an array of integers:

```
int getMaxIntArray(int *intArray, int numEl)
{
    int max = intArray[0];
    int i;

    for(i = 1; i < numEl; i++)
    {
        if(intArray[i] > max)
        {
            max = intArray[i];
        }
    }

    return max;
}
```

Case Study

- Suppose we want a function that return the maximum value from an array of `floating-point number`:

```
float getMaxFloatArray(float *floatArray, int numEl)
{
    float max = floatArray[0];
    int i;

    for(i = 1; i < numEl; i++)
    {
        if(floatArray[i] > max)
        {
            max = floatArray[i];
        }
    }

    return max;
}
```

Case Study

- How about from an array of string?

```
char * getMaxStringArray(char *stringArray[], int numEl)
{
    char *max = stringArray[0];
    int i;

    for(i = 1; i < numEl; i++)
    {
        if(strcmp(max, stringArray[i]) < 0)
        {
            max = stringArray[i];
        }
    }

    return max;
}
```

Note that we use `strcmp()` function.

Case Study

- Assume that we have a structure named `person` as follows:

```
struct person
{
    char name[100];
    int age;
    float height;
};
```

- If we have an array of `struct person`, and we need to get the maximum, we need to define what is mean by maximum, by name, by age, or by height.
- So, we need three functions for each of its component:

```
struct person getMaxSPArrayByName(struct person *spArray,
                                  int numEl) {...}
struct person getMaxSPArrayByAge(struct person *spArray,
                                 int numEl) {...}
struct person getMaxSPArrayByHeight(struct person *spArray,
                                    int numEl) {...}
```

Case Study

- Note that all six functions are pretty much the same
  - Same main functionality
  - Return different type
  - Take different types as arguments
- Ideally, this is not a good programming practice
- In Java, we use **Generic** or **Type Variables**
- Sadly, we do not have those in C.
- To solve this problem, we need to a function that
  1. takes various types as arguments
  2. returns value of various type

void Pointer

- Recall that a void pointer can be used to point to any type including arrays in C:

```
int x;
float y;
char z;
struct person p;

int a[10];
float b[20];
char c[30];
struct persion r[5];

void *v_ptr;

v_ptr = &x;
v_ptr = &y;
v_ptr = &c;
v_ptr = &p;

v_ptr = a;
v_ptr = b;
v_ptr = c;
v_ptr = r;
```

Various Type Argument

- If we want a function to be able to take an argument of any type, use type `void` pointer

```
int foo(void *arg) {...}
```

- In doing so, we can send any type to the function `foo()` by simply send the address

```
int x = 5;
float y = 2.2;
char c = ’A’;
int a[10];

foo(&x);
foo(&y);
foo(&c);
foo(a);
```

- **Remember**: `foo()` has no idea about the type it receives.

Various Return Type

- Similarly, a function can return various type by simply return a void pointer.

```
void * foo(void *arg) {...}
```

- The caller should know what to expect and cast it to the right type

```
int result;
int x[10];
...
result = (int *) foo(x);
```

How about our getMax...() function?

- So far, we have six getMax...() functions as follows:

```
int getMaxIntArray(int *intArray, int numEl);
float getMaxFloatArray(float *floatArray, int numEl);
char * getMaxStringArray(char *stringArray, int numEl);
struct person getMaxSPArrayByName(struct person *spArray, int numEl);
struct person getMaxSPArrayByAge(struct person *spArray, int numEl);
struct person getMaxSPArrayByHeight(struct person *spArray, int numEl);
```

- Those functions almost have the same implementation including:
  - the first argument is an array of some types,
  - the second argument is the number of elements in array, and
  - they return a value of some types.
- We can have one function that support all those by using void pointers:

```
void * getMaxFromArray(void *array, int numEl);
```

Note that getMaxFromArray() does not have any information about type it receives.

Implementing getMaxFromArray()

- Recall that functions getMax...() need to compare the initial maximum value with the rest of the array.

```
    int max = intArray[0];
    int i;

    for(i = 1; i < numEl; i++) {
        if(intArray[i] > max) {
            max = intArray[i];
        }
    }
```

- Using square brackets ([..]) to access elements in an array only work if we know the type of the array.
- Since we do not know the type, we need to know the size of each element
    - We need to know the offset of a specific element from its base address
- Thus, the caller must send the size of each element as an argument

Implementing getMaxFromArray()

- New signature:

```
void * getMaxFromArray(void *array, int numEl, int size);
```

- Caller need to pass the size of each element

```
int x[] = {...};
float y[] = {...};
char *str[] = {...};
struct person p[] = {...};

// Ignore return values for now

getMaxFromArray(x, 10, sizeof(int));
getMaxFromArray(y, 10, sizeof(float));
getMaxFromArray(str, 10, sizeof(char *));
getMaxFromArray(p, 10, sizeof(struct person));
```

- At this point, getMaxFromArray() knows how to traverse the array but it still does not know how to compare two elements of a given array.

Implementing getMaxFromArray()

- Add another argument that indicate the type is not enough
  - getMaxFromArray() should work with any type including newly create structure
  - You may not be the one who implement getMaxFromArray() to add more support types
- **Solution**: Send a function as an argument for getMaxFromArray() to use to compare two elements
- Requirements:
  - The function must takes two argument of type void *
  - It should return an integer where
    - 0: two arguments are equal
    - positive value: the first argument is larger
    - negative value: the first argument is smaller
- Final signature

```
void * getMaxFromArray(void *array, int numEl, int size
                       int (*compare)(void *, void *));
```

Implementing getMaxFromArray()

- Final signature:

```
void * getMaxFromArray(void *array, int numEl, int size
                       int (*compare)(void *, void *));
```

- Caller need to supply the following:
  - An array
  - Number of elements used in the array
  - The size of each element of the array
  - A function to be used to compare elements in the array

getMaxFromArray()

```
void * getMaxFromArray(void *array, int numEl, int size,
                       int (*compare)(void *first, void *second))
{
    void *max = array;
    int i;

    for(i = 1; i < numEl; i++)
    {
        void *temp = array + (i * size);

        if(compare(max,temp) < 0)
        {
            max = temp;
        }
    }

    return max;
}
```

Function to Compare Integer

- Suppose you want to use `getMaxFromArray()` with an array of integers
- Create a function to compare two integers:

```
int compareInt(void *first, void *second)
{
    return *((int *) first) - *((int *) second);
}
```

- How to use?

```
int intArray[] = {...};
...
int maxInt = *((int *) getMaxFromArray(intArray, 9, sizeof(int),
                                       compareInt));
```

Function to Compare struct person by Age

- Suppose you want to use getMaxFromArray() with an array of integers
- Create a function to compare two integers:

```
int compareSPAge(void *first, void *second)
{
    struct person p1 = *(struct person *) first;
    struct person p2 = *(struct person *) second;
    return p1.age - p2.age;
}
```

- How to use?

```
int personArray[] = {...};
...
struct person maxPersonAge = *((struct person *) getMaxFromArray(
        personArray, 4, sizeof(struct person), compareSPAge));
```