

C/C++ Function Pointer

Data Structures
C++ for C Coders

한동대학교 김영섭 교수
idebtor@gmail.com

Functions as pointers

- Function code is stored in memory
- Start of the function code or the address of a function is a “function pointer”
- Function pointer is “different” from other pointers since you do not allocate or deallocate memory with them
- Function pointers can be passed as arguments to other functions or return from functions

Why use function pointers?

- Efficiency
- Elegance
- Runtime binding
 - Determine sorting function based on type of data **at run time**
 - Eg: insertion sort for smaller data sets ($n < 100$)
 - Eg: Quicksort for large data sets ($n > 100000$)
 - Other sorting algorithms based on type of data set

Defining a function pointer

- For example,

```
int gcd(int x, int y) {  
    if (y == 0) return x;  
    return gcd(y, x % y);  
}
```

```
int main() {  
    int (*fn)(int, int);  
    fn = gcd;  
    int ans = fn(259, 111);  
    cout << ans << endl;  
}
```

Defining a function pointer

```
#include <iostream>
using namespace std;

int fun(int x, int y) {
    return x * 2 + y;
}

int foo(int x, int y) {
    return x + y * 2;
}

int add(int x, int y) {
    return x + y;
}
```

```
int main() {
    // using function pointer
    int (*fp) (int, int) = fun;
    cout << "fp(): " << fp(2,3) << endl;
    fp = foo;
    cout << "fp(): " << fp(2,3) << endl;
}
```

Arrays of Function Pointers

- C/C++ treats pointers to functions just like pointers to data therefore we can have arrays of pointers to functions
- This offers the possibility to select a function using an index.

Arrays of Function Pointers

```
#include <iostream>
using namespace std;

int fun(int x, int y) {
    return x * 2 + y;
}

int foo(int x, int y) {
    return x + y * 2;
}

int add(int x, int y) {
    return x + y;
}
```

```
int main() {
    // fp is an array of fp
    int (*fp[])(int, int) = {fun, foo, add};
    for (int i = 0; i < 3; i++)
        cout << "fp(" << i << "): "
              << fp[i](2, 3) << endl;;
}
```

qsort

function

qsort

<stdlib.h>

```
void qsort (void* base, size_t num, size_t size,  
           int (*compar)(const void*,const void*));
```

Sort elements of array

Sorts the *num* elements of the array pointed to by *base*, each element *size* bytes long, using the *compar* function to determine the order.

The sorting algorithm used by this function compares pairs of elements by calling the specified *compar* function with pointers to them as argument.

The function does not return any value, but modifies the content of the array pointed to by *base* reordering its elements as defined by *compar*.

The order of equivalent elements is undefined.

```
void qsort (void* base, size_t num, size_t size,  
           int (*compar) (const void*,const void*)) ;
```


using qsort

A



11 22 40 29 50 32 21 78 23

A is an array of integers.

Sort it using qsort with natural integer order

Write the compare function:

```
int (*intcomp)(const void* a, const void* b))
```

```
void qsort (void* base, size_t num, size_t size,  
            int (*compar)(const void*,const void*));
```

using qsort

A



"hat" "put" "ace" "bat" "cat"

A is an array of strings or char *'s.
Sort it using qsort with alphabetical order

Write the compare function:

```
int (*strcmp)(const void* a, const void* b))
```

```
void qsort (void* base, size_t num, size_t size,  
            int (*compar)(const void*,const void*));
```

using qsort

A



An array of node *'s

mat 10

cat 5

pot 7

A is an array of data structure such as node *'s.
Sort it using qsort with alphabetical order

Write the compare function to sort by name:

```
int (*nodecomp)(const void* a, const void* b))
```

```
void qsort (void* base, size_t num, size_t size,  
            int (*compar)(const void*,const void*));
```

qsort example

```
#include <iostream>
int values[] = { 4, 1, 7, 9, 2, 3 };

int compare (const void *a, const void *b) {
    return ( *(int*)a - *(int*)b );
}

int main () {
    int n = sizeof(values)/sizeof(values[0]);
    qsort (values, n, sizeof(int), compare);

    for (int i=0; i<n; i++)
        std::cout << values[i] << " ";
    return 0;
}
```

bubblesort

- **Function bubblesort takes a function pointer**
- **The argument in bubblesort for the function pointer:**
 - `int (*compare) (int a, int b)`
tells bubblesort to expect a pointer to a function that takes two ints and returns an int
- **If the parentheses were left out:**
 - `int *compare(int a, int b)`

bubblesort

- **Function bubblesort takes a function pointer**
- **The argument in bubblesort for the function pointer:**
 - `int (*compare) (int a, int b)`
tells bubblesort to expect a pointer to a function that takes two ints and returns an int
- **If the parentheses were left out:**
 - `int *compare(int a, int b)`
 - **Defines a function that receives two integers and returns a pointer to a int**

bubblesort example

```
void bubblesort(int *list, int n) {  
    int i, j, temp;  
  
    for (i = 0; i < n - 1; i++) {  
        for (j = 0; j < n - i - 1; j++)  
            if (list[j] > list[j + 1]) {  
                temp = list[j];  
                list[j] = list[j + 1];  
                list[j + 1] = temp;  
            }  
        }  
    }  
}
```

← **no function pointer used**

bubblesort example

```
void bubblesort(int *list, int n) {
    int i, j, temp;

    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++)
            if (list[j] > list[j + 1]) {
                temp = list[j];
                list[j] = list[j + 1];
                list[j + 1] = temp;
            }
    }
}
```

```
int main() {
    int list[] = { 3, 4, 1, 7, 9, 5 };
    int N=sizeof(list)/sizeof(list[0]);

    cout << "UNSORTED: " << endl;
    for (int i = 0; i < N; i++)
        cout << list[i] << " ";
    cout << endl;

    bubblesort(list, N);
    cout << "SORTED: " << endl;
    for (int i = 0; i < N; i++)
        cout << list[i] << " ";
    cout << endl;
}
```

no function pointer used

Use function pointer for ascending or descending sort

Use function pointer for ascending or descending sort

```
int main() {  
    int list[] = { 3, 4, 1, 7, 9, 5 };  
    int N=sizeof(list)/sizeof(list[0]);  
  
    cout << "UNSORTED: " << endl;  
    for (int i = 0; i < N; i++)  
        cout << list[i] << " ";  
    cout << endl;  
  
    bubblesort(list, N, ascending);  
    cout << "SORTED: " << endl;  
    for (int i = 0; i < N; i++)  
        cout << list[i] << " ";  
    cout << endl;  
}
```

Use function pointer for ascending or descending sort

```
int main() {  
    int list[] = { 3, 4, 1, 7, 9, 5 };  
    int N=sizeof(list)/sizeof(list[0]);  
  
    cout << "UNSORTED: " << endl;  
    for (int i = 0; i < N; i++)  
        cout << list[i] << " ";  
    cout << endl;  
  
    bubblesort(list, N, ascending);  
    cout << "SORTED: " << endl;  
    for (int i = 0; i < N; i++)  
        cout << list[i] << " ";  
    cout << endl;  
}
```

```
int ascending (int a, int b) {return a - b;}  
int descending(int a, int b) {return b - a;}
```

Use function pointer for ascending or descending sort

```
void bubblesort(int *list, int n,  
                int (*comp)(int a, int b)) {  
  
    int i, j, temp;  
  
    for (i = 0; i < n - 1; i++) {  
        for (j = 0; j < n - i - 1; j++)  
            if (comp(list[j], list[j + 1]) > 0) {  
                temp = list[j];  
                list[j] = list[j + 1];  
                list[j + 1] = temp;  
            }  
    }  
}
```

```
int ascending (int a, int b) {return a - b;}  
int descending(int a, int b) {return b - a;}
```

```
    { 3, 4, 1, 7, 9, 5 };  
    N = (int)list / sizeof(list[0]);
```

```
    printf("Sorted: " << endl;  
    for (i = 0; i < N; i++)  
        printf("%d ", list[i]);
```

```
    bubblesort(list, N, ascending);  
    printf("Sorted: " << endl;  
    for (i = 0; i < N; i++)  
        printf("%d ", list[i]);
```