C++ For C Coders 6

Data Structures
C++ for C Coders

한동대학교 김영섭교수
idebtor@gmail.com

function pointer
array of function pointers
function pointer as an argument

1. Function Pointer - Pointer to function

- Function pointer:
  - Function code is stored in memory
  - Function pointer is the start address of a function **code**, not data.

- It is the same kind since
  - it can be **passed as an argument** to other functions or can also be returned from a function.
  - we can have **an array of function pointers** just like data pointers.

- It is a different kind since
  - you do not allocate or deallocate memory with it.
  - It can be used as a **callback function**.

2. Declaring a function pointer

- Let's look at a simple example:

```
void (*foo)(int);
```

- **foo** is a pointer to a function taking one argument, an integer, and that returns **void**.
- It's as if you're declaring a function called "***foo**", which takes an **int** and returns **void**;
- Since ***foo** is a function, then **foo** must be a pointer to a function.
  - Similarly, a declaration like **int *x** can be read as ***x** is an **int**, so **x** must be a pointer to an **int.**

- For example,

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

int main() {
  cout << gcd(259, 111) << endl;

  int (*fp)(int, int) = gcd;

  cout << fp(259, 111) << endl;
}
```

3. Using function pointer

fps.cpp

```
#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() {
  cout << "fun(2,3): " << fun(2, 3) << endl;

  // the ampersand is optional
  int (*fp) (int, int) = fun;    // &fun
  cout << "fun(): " << fp(2, 3) << endl;

  fp = foo;
  cout << "foo(): " << fp(2, 3) << endl;

  fp = add;
  cout << "add(): " << fp(2, 3) << endl;
  cout << "add(): " << (*fp)(2, 3) << endl;
}
```

4. Using an Array 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.

4. Using an Array of Function Pointers

fps.cpp

```
#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() {
  // fps[] is an array of fp

  int                                  {fun, foo, add};
  int N =
  for (int i = 0; i < N; i++)
    cout << "fp(): " << fps[i](2,3) << endl;
}
```

4. Using an Array of Function Pointers

fps.cpp

```
#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() {
  // fps[] is an array of fp

  int (*fps[])(int, int) = {fun, foo, add};
  int N = sizeof(fps) / sizeof(fps[0]);
  for (int i = 0; i < N; i++)
    cout << "fp(): " << fps[i](2,3) << endl;
}
```

4. Using an Array of Function Pointers

fps.cpp

```
#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() {
  // fps[] is an array of fp

  int (*fps[])(int, int) = {fun, foo, add};
  int N = sizeof(fps) / sizeof(fps[0]);
  for (int i = 0; i < N; i++)
    cout << "fp(): " << fps[i](2,3) << endl;
}
```

How about using for-each loop?

4. Using an Array of Function Pointers

fps.cpp

```
#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() {
  // fps[] is an array of fp

  int (*fps[])(int, int) = {fun, foo, add};
  int N = sizeof(fps) / sizeof(fps[0]);
  for (int i = 0; i < N; i++)
    cout << "fp(): " << fps[i](2,3) << endl;
}
```

How about using for-each loop?

```
int main() {
  int (*fps[])(int, int) = {fun, foo, add};
  for (auto fp: fps)
    cout << "fp(): " << fp(2,3) << endl;
}
```

5. Function pointer as an argument

fps.cpp

- Implement `op_print()` that accepts one of three function pointers (`fun, foo, add`) and two int arguments. Then, it prints the result of the operation.

```
void op_print(_____________________, int a, int b) {
    cout << "op(): " << op(a, b) << endl;
}

int main() {
  int (*fps[])(int, int) = {fun, foo, add};
  int N = ____________________________________;
  for (int i = 0; i < N; i++)
     op_print(_________________, 2, 3);
}
```

Complete the code for yourself

5. Function pointer as an argument

fps.cpp

- Implement `op_print()` that accepts one of three function pointers (`fun, foo, add`) and two int arguments. Then, it prints the result of the operation.

```
void op_print(_____________________, int a, int b) {
    cout << "op(): " << op(a, b) << endl;
}

int main() {
  int (*fps[])(int, int) = {fun, foo, add};

  for ( ____________________________ )
     op_print(_________________, 2, 3);
}
```

Use for-each loop

6. qsort example

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

- The `qsort()` function in C/C++ sorts a given array using Quicksort algorithm.
- The function is defined in `<cstdlib>` header file in C++.
- The `qsort()` function sorts the given array pointed by `base` in ascending order. The array contains `num` elements, each of `size` bytes.
- The function pointed by `compare` is used to compare two elements of the array. This function modifies the content of the array itself sorted.

6. qsort example - version 1

qsort.cpp

```
#include <iostream>
int comp_int (const void *a, const void *b) {
  return ( *(int*)a - *(int*)b );
}

int main () {
  int values[] = { 2, 8, 1, 9, 5 };
  int n = 5;
  qsort (values, n, sizeof(int), comp_int);

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

6. qsort example - version 2: NMN

qsort.cpp

```
#include <iostream>
int comp_int (const void *a, const void *b) {
  return ( *(int*)a - *(int*)b );
}

int main () {
  int values[] = { 2, 8, 1, 9, 5 };
  int n =
  qsort (values, n, sizeof(          ), comp_int);

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

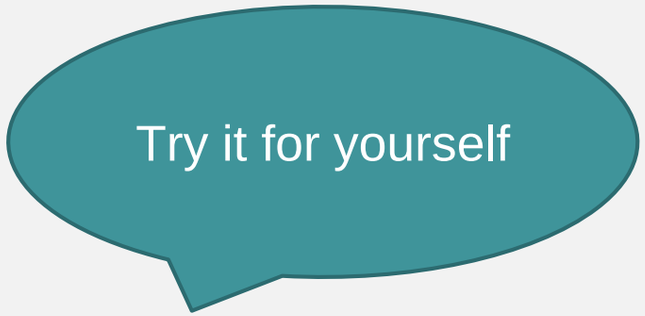
6. qsort example - version 3: char array

qsort.cpp

```
#include <iostream>
int comp_int (const void *a, const void *b) {
  return ( *(int*)a - *(int*)b );
}

int main () {
  char values[] = { 'n', 'i', 'b', 'c' };
  int n = sizeof(values) / sizeof(values[0]);
  qsort (values, n, sizeof(values[0]), comp_int);

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



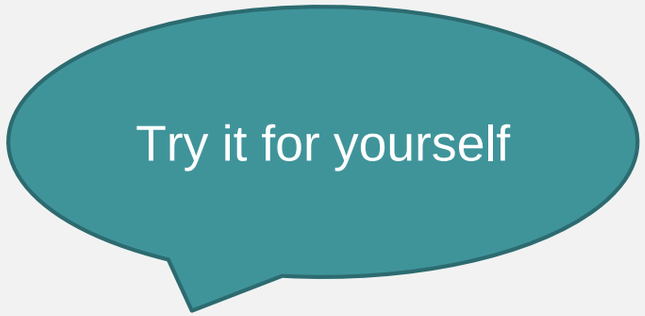
6. qsort example - version 4: descending

qsort.cpp

```
#include <iostream>
int comp_int (const void *a, const void *b) {
  return ( *(int*)a - *(int*)b );
}

int main () {
  int values[] = { 2, 8, 1, 9, 5 };
  int n = sizeof(values) / sizeof(values[0]);
  qsort (values, n, sizeof(values[0]), comp_int);

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




```
for (auto value: values)
   std::cout << value << " ";
```

C++ For C Coders 6

Data Structures
C++ for C Coders

한동대학교 김영섭교수
idebtor@gmail.com

function pointer
array of function pointers
function pointer as an argument