Lecture 4
Arrays and Pointers

array, dynamic array, pointer vs reference

Prof. Hyeong-Seok Ko
Seoul National University
Graphics & Media Lab

Contents

- Array (4.1, 4.4)
- Dynamically Allocating Array (4.3.1)
- Pointer, Reference (4.2, 2.5)

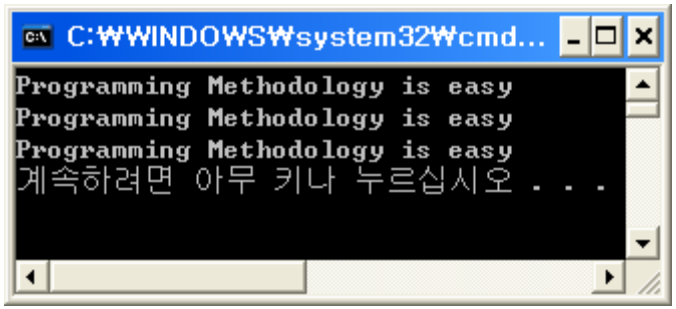
Array

- Container of objects of a single data type
- Fixed size

```
int iarray[3] = { 5,3,4 };
int marray[2][3] = { {0,1,2}, {3,4,5} };   // multi-dimensioned array

iarray[2] = 9;
marray[1][2] = 8;
```

Pointer



- A **pointer** is a variable
  - that holds the address of an object
- A **pointer** enables indirect access to an object.

```
#include <iostream>
#include <string>

void main() {
   int *m; m = new int; *m = 7;
   int * n = new int; *n = *m + 1;
   int* k; *k = 6;                    // a frequently made mistake

   std::string str = "Programming Methodology is easy";
   std::string * ptr = &str;          // ptr holds the address of str
     // What happens if ptr = str; ?  It causes a compile error
   std::string * ptr_a[10];           /* ptr_a is an array of 10 pointers
                                         of std::string */
   ptr_a[0] = &str;

   std::cout << str << std::endl;
   std::cout << *ptr << std::endl;   // indirect access to str
   std::cout << *ptr_a[0]<< std::endl;
}
```

Heap memory allocation

Pointer Arithmetic

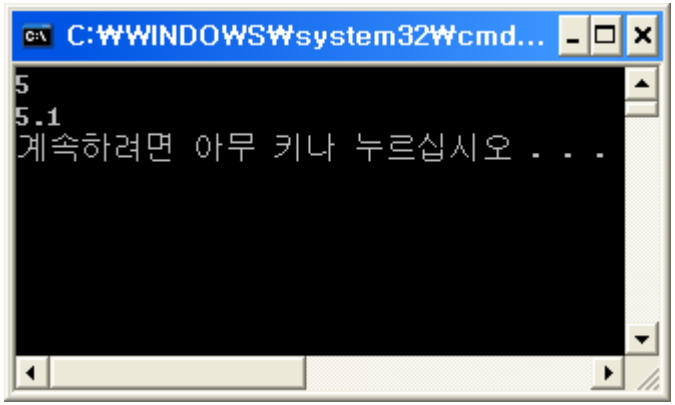
- Adding an integer n to a pointer variable x does not produce the address displaced from n **bytes** from x, but produces the address displaced n **elements** of the data type from x.

```
#include <iostream>

void main() {
    int ia[] = {0,1,2,3,4,5,6};
    int * ptr_i = &ia[3];

    double da[] = {0.1,1.1,2.1,3.1,4.1,5.1,6.1};
    double * ptr_d = &da[3];

    std::cout << *(ptr_i + 2) << std::endl;
    std::cout << *(ptr_d + 2) << std::endl;
}
```



Usage of Const Qualifier with Pointers

- Pointers and the `const` Qualifier
  - `const` qualifier can apply to pointers in two ways.
    - Pointers to `const` Objects
    - `const` Pointers

```
#include <iostream>

void main() {
   const double d0 = 10.0;
   const double * ptr_c = &d0;

   *ptr_c = 20.0;    // Compilation error
                     // You cannot assign to a variable that is const

   double * ptr = &d0;    // Compilation error
                          // cannot convert from 'const double *' to 'double *'
}
```

Pointer to const object

```
#include <iostream>

void main() {
   double d0 = 10.0, d1 = 20.0;
   double * const ptr_c = &d0;

   ptr_c = &d1;      // Compilation error
                     // ptr_c is a constant pointer
}
```

const Pointer

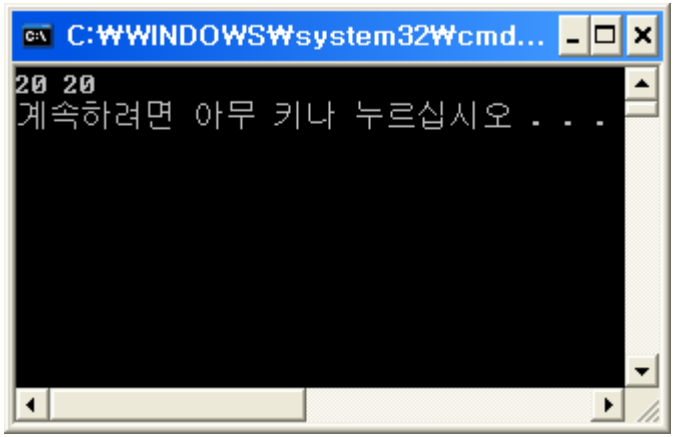
Reference

- Reference
  - A **reference** serves as an alternative name (alias) for an object.

```
#include <iostream>

void main() {
    double d0 = 10.0;
    double & d_ref = d0;  // d_ref is an alternative name for d0

    d_ref = 20.0;
    std::cout << d0 << " " << d_ref << std::endl;
}
```



Pointer vs. Reference

- A reference must refer a pre-existing object.
  - To define a reference without initializing it is an error.

```
#include <iostream>

void main() {
   double * d_ptr;  // No problem
   double & d_ref;  // Compilation Error
                    // reference must be initialized
}
```

- Note that assignment is done differently.

```
#include <iostream>

void main() {
   double d0 = 10.0, d1 = 10.0;
   double * d_ptr = &d0;
   double & d_ref = d0;

   d_ptr = &d1;  // d_ptr now points to d1
   d_ref = 20.0; // assign 20.0 to d0
}
```

Dynamically Allocating Array

- Allocate an array dynamically at run time

```
int n = 3, m0 = 2, m1 = 3;

int * iarray = new int[n];          // allocating
int ** marray = new int*[m0];
for(int i=0;i<m0;++i)
   marray[i] = new int[m1];

delete [] iarray;                   // de-allocating
for(int i=0;i<m0;++i)
   delete [] marray[i];
delete [] marray;
```