Topic 7
References & Pointers

Department of
Engineering Physics
University of Gaziantep

Course web page
www.gantep.edu.tr/~bingul/ep241

Sep 2013
Variables and Memory Addresses

Computer memory can be considered as a very large array of bytes.

For example, a computer with 1 GB of RAM actually contains an array of
1024 x 1024 x 1024 = 1,073,741,824 B.

\[0 = 0x00000000\]
\[1,073,741,824 = 0x3fffffff\]
When a variable is declared and assigned to a value four fundamental attributes associated with it:

- its name
- its type
- its value (content)
- its address

e.g.

```c
int n = 25;
```

![Variable memory diagram]
In C/C++ the address operator (&) returns the memory address of a variable.

```cpp
int main(){
    int n = 33;
    cout << " n = " << n << endl;
    cout << "&n = " << &n << endl;
}
```

n = 33
&n = 0x0024fdf0
References

- The **reference** is an **alias**, a **synonym** for a variable.
- It is decelerated by using the **reference operator** `&`.

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

int main(){
    int n = 33;
    int &r = n; // r is a reference for n

    cout << n << " " << r << endl;
    --n;
    cout << n << " " << r << endl;
    r *= 2;
    cout << n << " " << r << endl;

    cout << &n << " " << &r << endl;
    return 0;
}
```

```
33 33
32 32
64 64
0xbfdd8ad4 0xbfdd8ad4
```

```
0xbfdd8ad4
n,r
33
int
```
```cpp
#include <iostream>
using namespace std;

void takas(double &x, double &y){
    double z;
    z = x;
    x = y;
    y = z;
}

int main(){
    double a = 11.1, b = 22.2;

    cout << "a b : " << a << " " << b << endl;
    takas(a, b);
    cout << "a b : " << a << " " << b << endl;
}
```

a b: 11.1  22.2
a b: 22.2  11.1
Pointers

- The address operator returns the memory address of a variable.
- We can store the address in another variable, called *pointer*.

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

int main()
{
    int n = 33;
    int* p = &n; // p holds the address of n
    cout << "n = " << n << endl;
    cout << "&n = " << &n << endl;
    cout << "p = " << p << endl;
    cout << "&p = " << &p << endl;
    cout << "*p = " << *p << endl;
}
```

```
0xbfdd8ad4
n 33
int

0xbfdd8ad0
p 0xbfdd8ad4
int*

n = 33
&n = 0xbfdd8ad4
p = 0xbfdd8ad4
&p = 0xbffafad0
*p = 33
```
```cpp
#include <iostream>
using namespace std;

void takas(double *x, double *y){
    double z;
    z  = *x;
    *x = *y;
    *y = z;
}

int main(){
    double a = 11.1,  b = 22.2;

    cout << "a b : " << a << " " << b << endl;
    takas(&a, &b);

    cout << "a b : " << a << " " << b << endl;
}
```

```
a b: 11.1  22.2
a b: 22.2  11.1
```
Pointers and Arrays

- The name of an array is the address of its first element.
- The array name is a constant pointer.

```c
float numbers[20];
float *ptr = &numbers[0];  // valid
```

The following assignments are equivalent:

```c
numbers[4] = 25.8;
*(ptr+4) = 25.8;
```
Dynamic Memory Management

The declaration:

```c
double mass[10];   // Array size define at compile-time
```

Alternatively we can use a named constant;

```c
const int n = 10;
double mass[n];   // Array size define at compile-time
```

Note that “Standard C++” Array size defined at run-time FORBIDDEN!

```c
int n;
cin >> n;
double mass[n];
```

* * * This type of arrays are called Static Arrays * * *

Your compiler might allow you to do this, but it is best to use only standard C++ features so that your program can be compiled on any platform that has a standard C++ compiler.
- C++ provides run-time or **dynamic arrays** for which memory is allocated during execution.
- To allocate memory dynamically at run-time we use `new` operator.

General form:

```c
pointer = new type; // for single element
```

```c
pointer = new type [number_of_elements];
```

For example, to request a 10 element block of type `int` dynamically, we can use

```c
    int * mass;
    mass = new int [10];
```

or

```c
    int * mass = new int [10];
```
The `delete` operator reverses the action of the `new` operator, that is it frees the memory allocated by the `new` operator.

Its form is:

```c++
delete pointer;  // single element
delete [] pointer;  // a block of elements
```

e.g.

```c++
delete [] mass;
```
int main (){
    double *x, mean, s;
    int i, n;

    while(true){
        cout << "How many elements: "; cin >> n;
        if(n<=0) break;

        x = new double[n];
        s = 0.0;
        cout << "Input elements: ";
        for(i = 0; i<n; i++){
            cin >> x[i];
            s += x[i];
        }

        mean = s/n;
        cout << "Mean = " << mean << endl;
        delete [] x;
    }
} // main
Sample output of the previous program:

<table>
<thead>
<tr>
<th>How many elements:</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input elements:</td>
<td>1 2 3</td>
</tr>
<tr>
<td>Mean =</td>
<td>2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How many elements:</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input elements:</td>
<td>2 4 5 9 1 0</td>
</tr>
<tr>
<td>Mean =</td>
<td>3.5</td>
</tr>
</tbody>
</table>

| How many elements: | 0     |