Advanced Programming
(BETC 1353)

Week 2: Pointers (Part 1)

Dr. Abdul Kadir

abdulkadir@utem.edu.my

Learning Outcome

- To learn the basic pointers and all pointer operators
- To be able to use pointers for passing arguments to functions using reference

Benefit of Pointers

- Pointers allows changing the content of arguments in calling functions

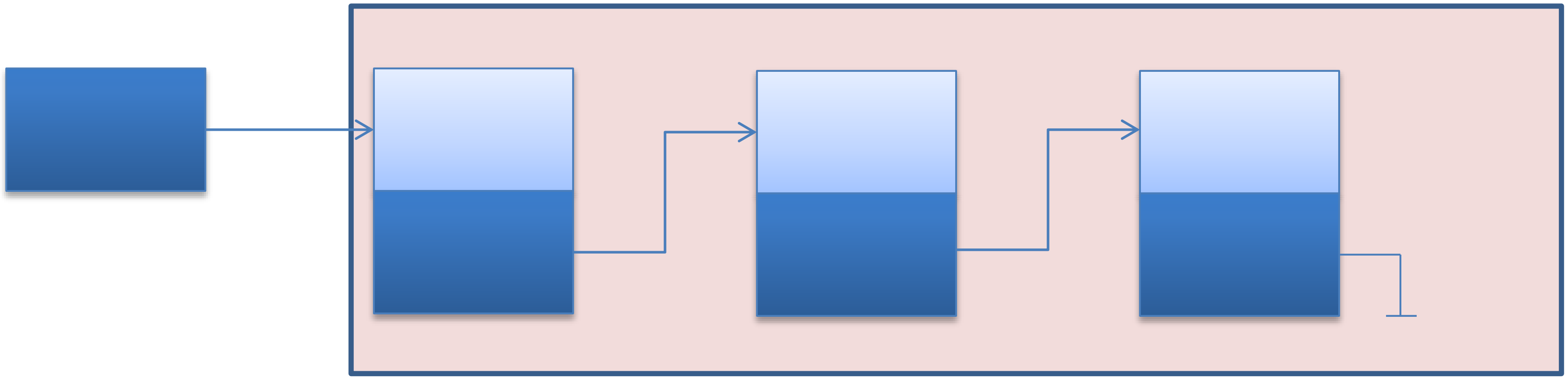
```
int main()
{
   int x = 5;
   increase(&x);

   // Now x is 6
}

int increase(int *x)
{
   *x = *x + 1;
}
```

Benefit of Pointers

- Useful for data structures, because pointers allows to point the dynamic data (added or inserted anytime in the computer memory)

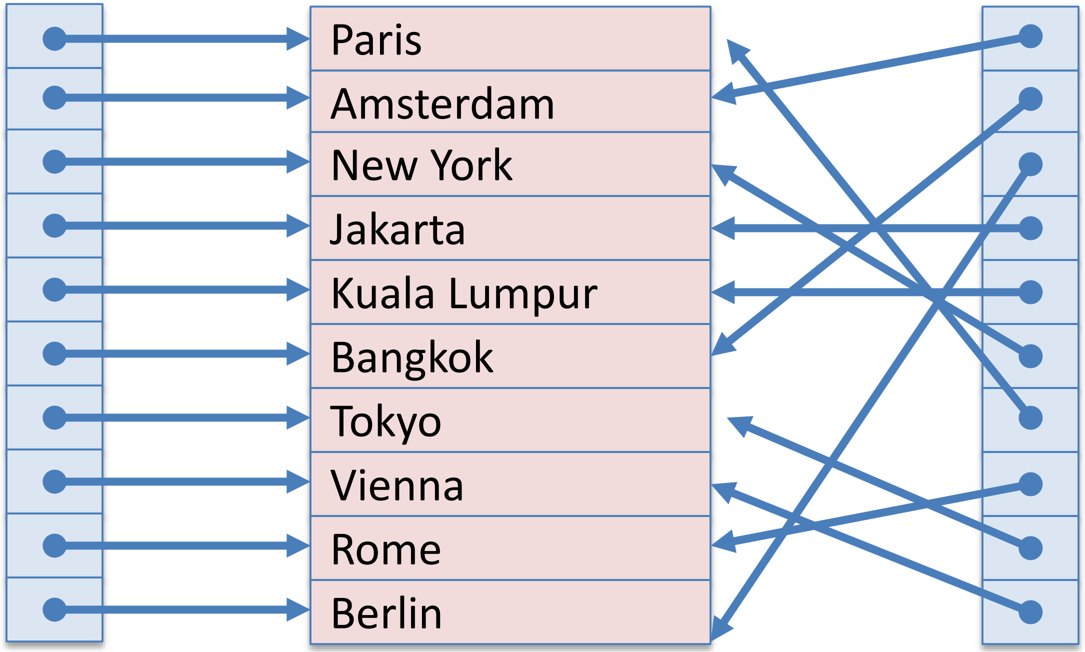


🟦 Pointers

⬜ Non-pointers

Benefit of Pointers

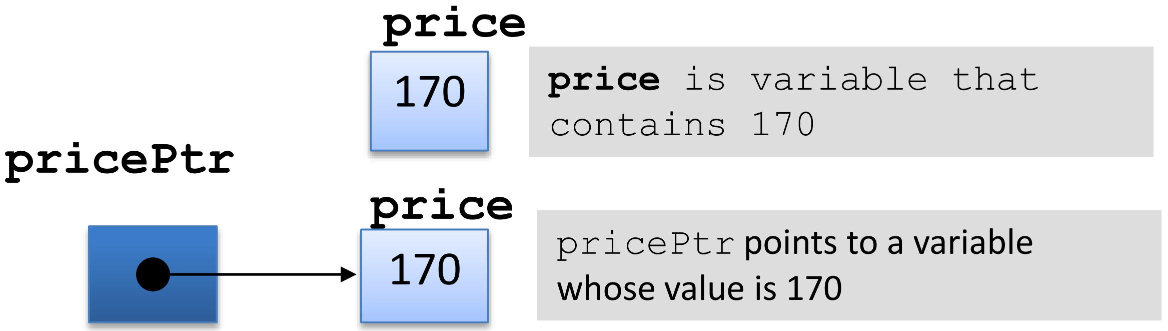
- Fast operation in sorting data because it does not need to swap the actual data -> Just swap the pointers



Pointer Variable Definition and Initialization

- Pointers contain address of a variable
- Their content can be changed to point another variable anytime.

- Example:



price is variable that contains 170

pricePtr points to a variable whose value is 170

- As a consequence, the content of price can be change directly using price or indirectly using pricePtr

Declaration

- Pointer declarations
  - use **`*`** in front of the variable names to declare pointer variables

```
        Syntax: datatype *varPtr;
```

  - **e.g.**

```
int *myPtr;

or

int* myPtr;
```

  - Data type **`int`** indicates that **`myPtr`** can be used to hold the address of an integer variable
  - is read as ”`myPtr is a pointer to int`” or “`myPtr points to an object of a type int`”

Declaration

- Multiple pointers can be declared in a statement
- A symbol of * is needed in front of each pointer variable
- For example:

```
int *ptrA, *ptrB;
```

is same as

```
int *ptrA, *ptrB;
```

Self-Evaluation

- Notice the following declaration:

```
char *x, y;
```

- Is `x` a pointer?
- Is `y` a pointer?

Declaration

- Pointer declarations
  - Can declare pointers to any data type
    - **e.g.** `float *discountPtr;`
  - By default, pointers are not initialized
  - Initializing the pointers can be done by giving **0**, **NULL**, or an address
    - **0** or **NULL** means points to nothing (**NULL** preferred)
    - For example:

```
float *discountPtr = 0;
```

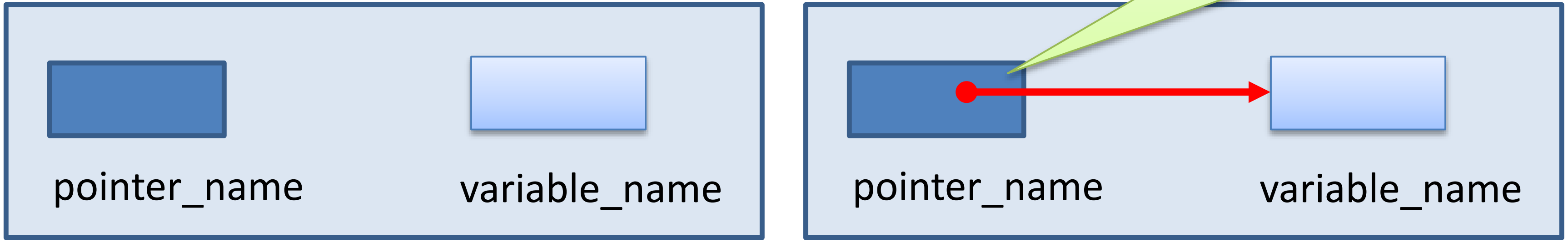
  - Rule of thumb: Always initialize pointers to prevent unexpected results

Pointer Operators

- **&** (address operator)
  - Returns the address of the operand
  - Syntax:

pointer_name = &variable_name;

Its content is the address of *variable_name*



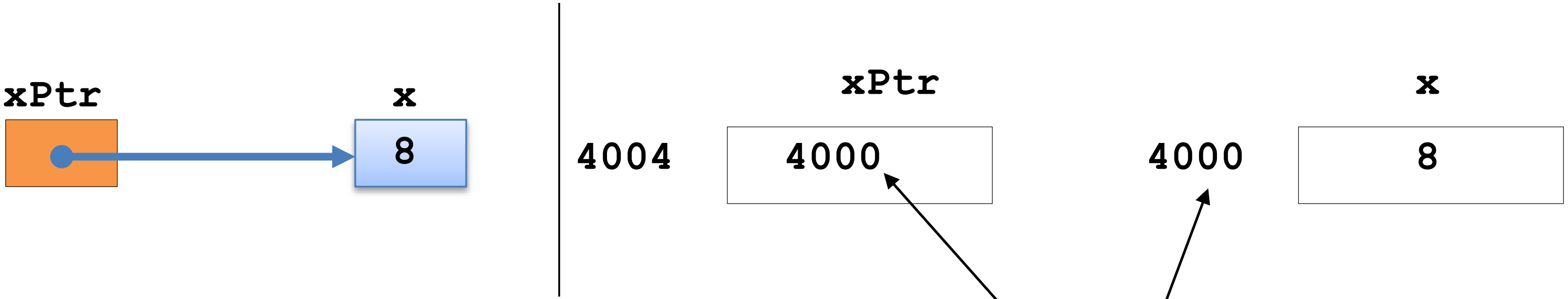
Before *pointer_name = &variable_name;* After *pointer_name = &variable_name;*

Pointer Operators

- **Example:**

```
int x = 8;
int *xPtr;
xPtr = &x;
```

xPtr gets address of **x**
xPtr "points to" **x**



Address of **x** is value of **xPtr**

Pointer Operators

- ***** (indirection/dereferencing operator)
  - Returns the value of the object pointed by the pointer
  - **`*xPtr`** returns the value of **`y`** (because **`yptr`** points to **`y`**)
  - e.g.

```
cout << x;
cout << *xPtr;
```

    - The output is 8

```
cout << *xPtr + 4;
```

The content of x + 4

    - The output is 12

  - ***** can be used for assignment

```
*xPtr = 7;  // Now, x is 7
```

Example

```
int main()
{
  float cost;
  float *costPtr;

  cost = 56.5;
  costPtr = &cost;

  value = *balptr;
  cout<< “Cost is: " << *costPtr << endl;
  cout<< “The address of cost is: ” << costPtr <<endl;

  return 0;
}
```

What is the output of this line?

Exercise 1

```
#include <iostream>

using namespace std;

int main()
{
    char ch_a = ‘A’;
    char ch_b = ‘Z’;
    char* ptr;
    char tmp;

    ptr = ch_a;
    tmp = *ptr;

    …

    return 0;
}
```

- Suppose, the memory addresses of the four variables are as follows:

❑ ch_a	: 0x28feec
❑ ch_b	: 0x28fee8
❑ ptr	: 0x28fee4
❑ tmp	: 0x28fee0

- What are the contents of ch_a, ch_b, ptr, and tmp before return 0; is executed?

Exercise 2

- What is the output for this example?

```
#include <iostream>

int main()
{
    char x = ’A’, y = ’B’;
    char *p1, *p2, *temp;
    p1 = &x;
    p2 = &y;
    temp = p1;
    p1 = p2;
    p2 = temp;
    cout << *p1 << ” “ << *p2;
}
```

Calling Functions by Reference

- Call by reference using pointer arguments
  - Passing the address of the argument using **&** operator
  - Allows us to change the content of the variable
- ***** operator
  - To allow us to change the content of variable outside the function in a function

```
void increase(int *num)
 {
    *num = *num + 1;
 }
```

parameter

  - ***num** means “pointed by num”

Example

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

void exchange(int *a, int *b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int m = 77, n = 88;
    cout << "Original values: " << m << " " << n << endl;
    exchange(&m, &n);
    cout << "After swap(): " << m << " " << n << endl;
    return 0;
}
```

Output:

```
Original values: m=77 n= 88
After swap(): m=88 n=77
```

What Happen when Reference is not used?

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

void exchange(int a, int b)
{
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int m = 77, n = 88;
    cout << "Original values: " << m << " " << n << endl;
    exchange(m, n);
    cout << " "After swap(): " << m << " " << n << endl;
    return 0;
}
```

<u>Output:</u>

```
Original values: m=77 n= 88
After swap(): m=77 n=88
```

Exercise 3 – What is the output of the program?

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

int main()
{
    int *pC, pD;
    int c = 78;
    int d = 34;

    pC = &c;
    pD = &d;

    cout << *pC << " "
     << *pD << endl;

    pD = pC;
    cout << *pC << " "
         << *pD << endl;

    *pD = *pC + 3;
    cout << *pC << " "
          << *pD << endl;

    return 0;
}
```