Advanced Programming (BETC 1353)

Week 3: Pointers (Part 2)

Dr. Abdul Kadir

abdulkadir@utem.edu.my

Learning Outcomes:

- Able to describe the concept of pointer expression and pointer arithmetic
- Able to explain the relationship between pointers and arrays
- Able to use arrays of pointers
- To apply pointers for sorting data using Bubble sort

The Relationship Between Pointers and Arrays

- Arrays and pointers have closely relationship
  - Array name is basically a constant pointer
  - In other words, the array name is an address
  - It is possible to use pointers to perform any array subscripting operations

The Relationship Between Pointers and Arrays

- For examples:
  Declare an integer array **`x[5]`** and an integer pointer **`xPtr`**

```
int x[5];
int *xPtr;
```

- To make the pointer points to the array:

```
xPtr = x;
```

  - In front of `b`, no & because `b` is an address (array name is an address)
  - It is equivalent to:

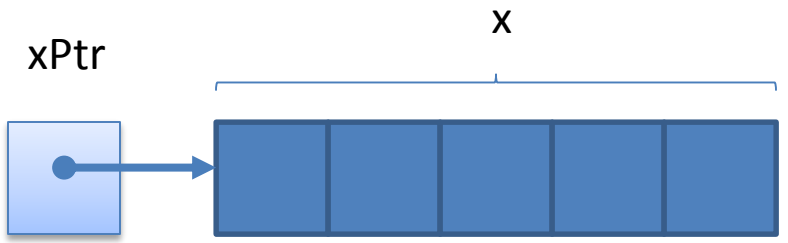
```
xPtr = &x[0];
```

The Relationship Between Pointers and Arrays (Cont..)

- After

```
xPtr = x;
```

xPtr point to x.

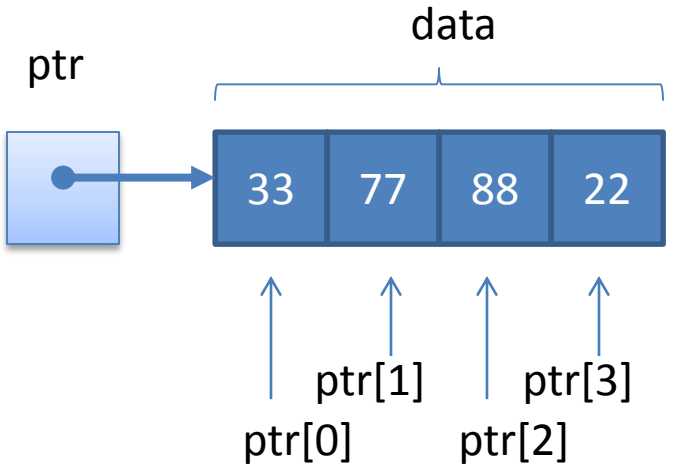


- Then, you can use the notation of xPtr[index] to access any element in the array x.
- For example: xPtr[0] = 45; to assign 45 to the first element in x

The Relationship Between Pointers and Arrays – An Example

- A pointer can be used to point an array
- Example:

```
#include <iostream>

using namespace std;

int main()
{
    int data[] = {33, 77, 88, 22};
    int *ptr;

    ptr = data; // No & required

    for (int j = 0; j < 4; j++)
       cout << ptr[j] << endl;

    return 0;
}
```

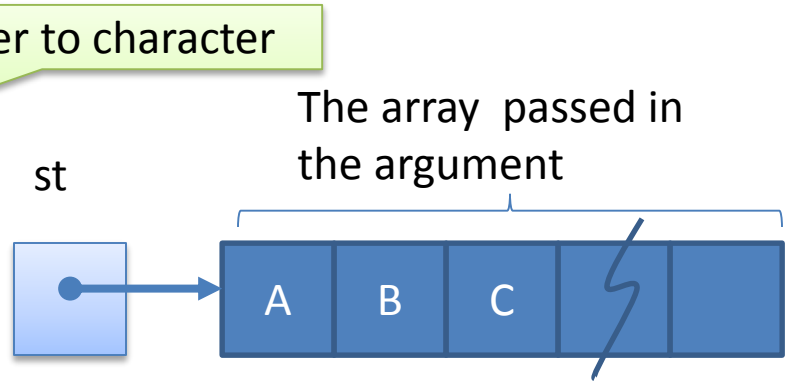


The pointer can be used to access each element in the array

Pointers and Strings

- A string is stored in an array of characters
- Therefore, you can used operations discussed in several previous slides
- Example of a pointer and a string is illustrated in the following function:

```
int numOfChars(char *st)
{
    int index = 0;
    while (st[index] != 0)
       index++;

    return index;
}
```



Pointers and Strings – An Example

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

int main()
{
   char text = ”Test..123!”;

   cout << numOfChars(text) << endl;

   return 0;
}

int numOfChars(char *st)
{
    int index = 0;
    while (st[index] != 0)
       index++;

    return index;
}
```

Call the function by passing text

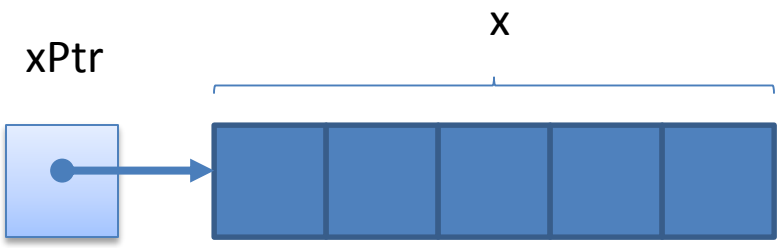
Exercise 4

- Write a function that receive a string and display the string in the reverse. For example, if the string of argument is “ABCDEF”, then the result is

  FEDCBA

Pointers and Arrays

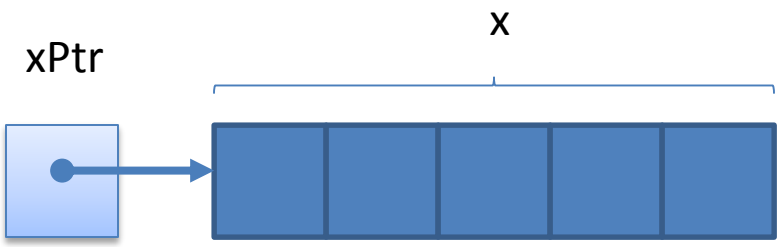
- Element `x[2]`
  - You can access it with:
    `*(xPtr + 2)`



  - In this case, **2** is the offset to the pointer. It is called **pointer/offset notation**
  - Generally, `x[n]` can be accessed using `*( xPtr + n )` where **n** is the index of the array

Pointers and Arrays

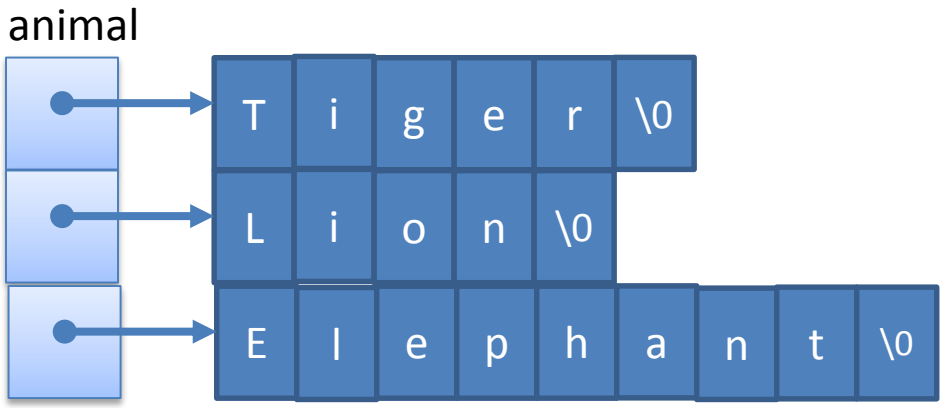
– Element `x[2]`
  - You can access it by
    `xPtr[2]`
  - It is called **pointer/subscript notation**
  - **So,** `xPtr[2]` is same as `x[2]`
  - The other approach to access it is
    `*(x+2)`



Arrays of Pointers

- Arrays can contain pointers
- For example: an array of strings

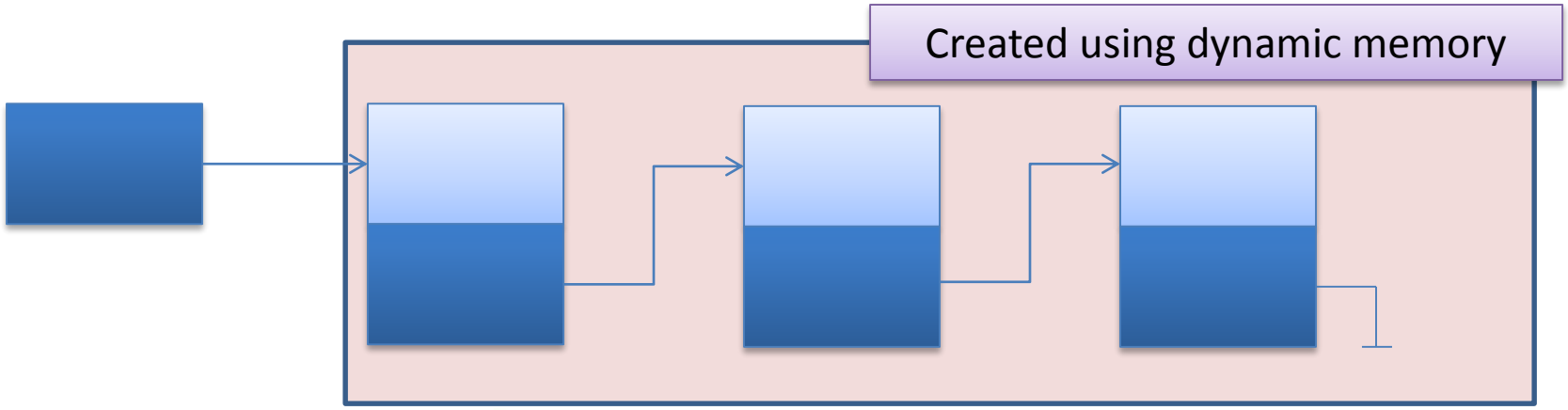
```
char *animal[3] =
{
    "Tiger",
    "Lion",
    "Elephant"
};
```



- The array animal has fixed size, but not for strings

Dynamic Memory

- Dynamic memory means memory that can be allocated or deallocated at run time
- It allows to create or delete dynamic data (variables) when the program run
- It is useful for creating data structures such as linked lists or queues

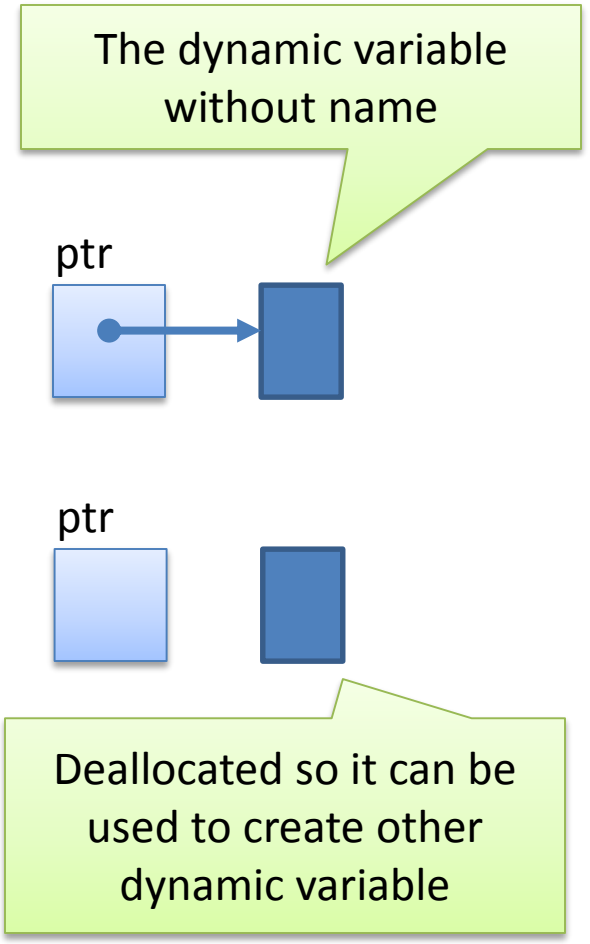


Dynamic Memory

- Two operators for allocating and deallocating memory:
  - new : for creating a dynamic variable

    Example: int *ptr = new int;
  - delete: for deleting a dynamic variable

    Example: delete ptr;
- The use of new and delete will be discussed in other chapter



Pointer Arithmetic Rules

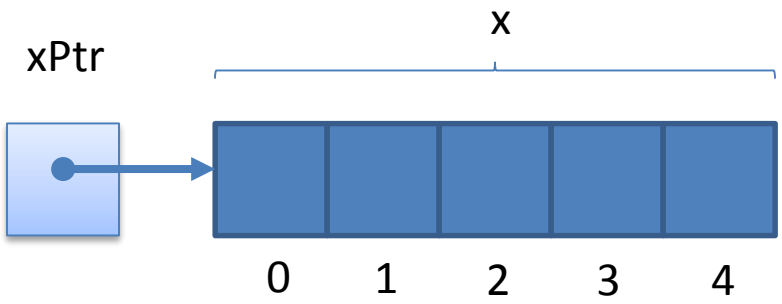
- Four basic arithmetic operators are available to be used on pointers: ++,--,+,-
- Examples:

```
ptr++;
ptr--;
ptr = ptr + 1;
ptr = ptr - 1;
```

Pointer Expressions and Pointer Arithmetic

- Suppose, there is a declaration as follows:

```
int x[5];
int *xPtr;
```



- **xPtr** points to the address of the first element **x[0]** after :

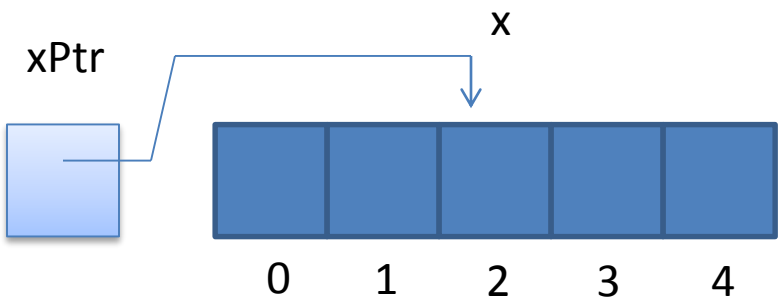
```
xPtr = x;
```

OR

```
vPtr = &x[0];
```

Pointer Expressions and Pointer Arithmetic

– Performing expression such as follows is allowed:

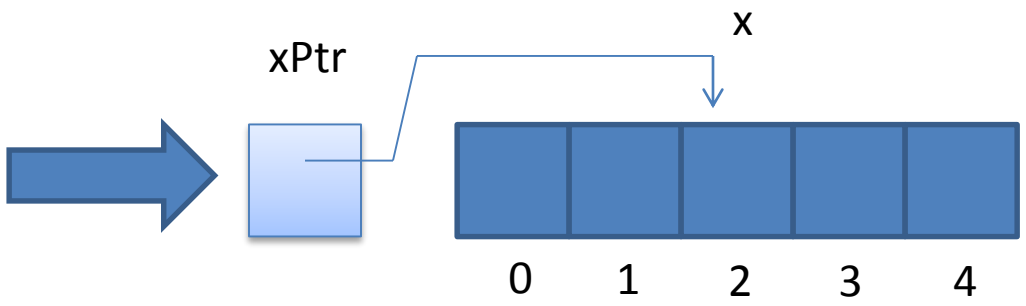



```
xPtr += 2;
```

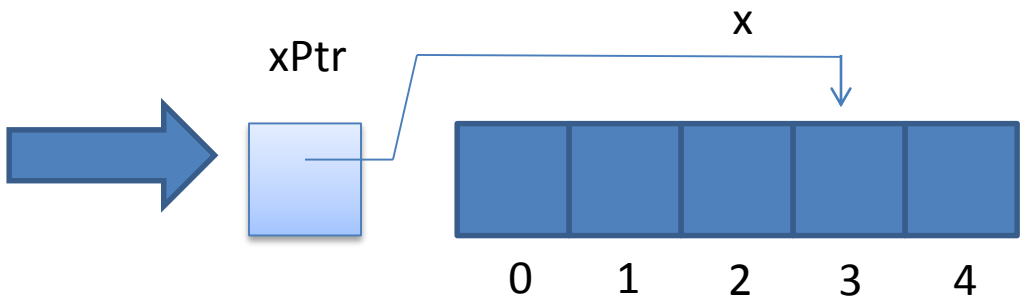
– In this case, xPtr will point the third element of x

Pointer Expressions and Pointer Arithmetic

– Before xPtr++;



– After xPtr++;



Example: Application of Pointer for Sorting Data Using Bubble Sort

- Usually, bubble sort are implemented by using array

- However, we will implement it using pointers
- Pointers are used to
  - Display the data
  - Swap to data
  - Handle sorting

Example : Bubble Sort

```
#include <iostream>
#include <iomanip>

const int MAX_DATA = 12;

using namespace std;

void displayData(int *data, int num);
void swap(int *x, int *y);
void bubbleSort( int *data, int num);

int main()
{
    int data[MAX_DATA] =
       {6, 8, 2, 1, 2, 67, 12, 34, 7, 45, 89, 21};

    cout << "Before sorting:"  << endl;
    displayData(data, MAX_DATA);

    bubbleSort(data, MAX_DATA);
    cout << "After sorting:"  << endl;
    displayData(data, MAX_DATA);

    return 0;
}
```

```
void displayData(int *data, int num)
{
    for (int j = 0; j < num; j++ )
        cout << setw(4) << data[j];

    cout << endl;
}

void bubbleSort( int *data, int num)
{
    for (int phase = 0; phase < num - 1; phase++)
        for (int j = 0; j < num - 1; j++)
            if (data[j] > data[j + 1])
                swap(&data[j], &data[j + 1]);
}

void swap(int *x, int *y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
```