

Class & Pointer Basics

Lecture 4

Professor Peter Cheung
EEE, Imperial College

- In this lecture*, we will
 - Examine Class Objects and Structure Objects in C++
 - Find out what are pointers
 - Find out the relationship between pointers and arrays when passed as parameters

* Savitch Chapters 6 & 12.1

Classes

- A class is a collection of related data identified as a single unit.
- Each element in a class is called a data member.
- After it has been declared, an instance of that class can be created for use in a program.

- Example of a class declaration:

```
Class TEmployee
{
    char FirstName[10];
    char LastName[20];
    double TotalHours;
    double HourlySalary;
    double WeeklySalary;
};
```

Accessing members in a class

```
Class TEmployee
{
    char FirstName[10];
    char LastName[20];
    double TotalHours;
    double HourlySalary;
    double WeeklySalary;
};
```

```
TEmployee FullTime;
double WeeklySalary;
```

```
FullTime.FirstName = "Chester";
FullTime.LastName = "Stanley";
FullTime.TotalHours = 42.00;
FullTime.HourlySalary = 10.63;
```

```
WeeklySalary = FullTime.TotalHours * FullTime.HourlySalary;
```

Note that WeeklySalary here is NOT part of the data structure

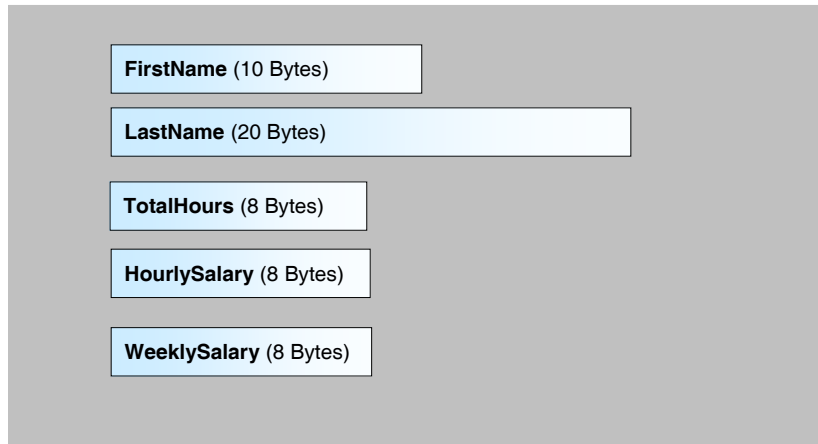
Quick Initialisation

- You can initialise a class object as it is declared:

```
TEmployee FullTime = {
    "Chester",
    "Stanley",
    42.0,
    10.63,
    0
};
```

You have to remember what order your data elements were declared

Object in memory



Private vs Public

- By default, everything inside a class object (i.e. all the data members) are **private** to the module that the object is declared. It is not visible outside.
- In order to make its members visible, you have to explicitly declare them as **public**.
- In other words, you can control the **scope** of the variable.

```

Class TEmployee
{
    public:
        char FirstName[10];
        char LastName[20];
        double TotalHours;
        double HourlySalary;
        double WeeklySalary;
    private:
};

```

Structures vs Classes

- In C++, structures and classes are very similar with one exceptions. Members within a structure are public by default. You need to explicitly make them private.
- The opposite is true with a class.
- Example of a structure is:

```

const Double PI = 3.14159;
//-----
struct TDoughnut
{
    string Name;
    Single Radius;
    Single radius;
    Single Area;
    Single CreameArea;
    Single Volume;
};

TDoughnut Original;

Original.Name = "Glazed";
Original.Radius = 12.50;
Original.radius = 3.75;
Original.Area = 4 * PI * Original.Radius * Original.radius;
Original.CreameArea = 2 * Original.Area / 5;
Original.Volume = 2 * PI * PI * Original.Radius *
    Original.radius * Original.radius;

```

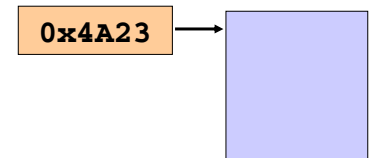
What is a pointer?

- An object (such as a class object or structure object) stored in memory can be referred to in two ways:
 - 1. The object's **content** itself (e.g. the value of an integer variable)
 - 2. The **location** at which the object is stored (e.g. the memory address of the integer variable).
- A **pointer** is a variable whose value represents the location (or address) of another object.
- Pointer objects are defined in conjunction with the unary indirection operator ***** (also known as the **dereferencing** operator).

```

int *iPtr = 0;
char *cPtr = 0;
float *fPtr = 0;

```

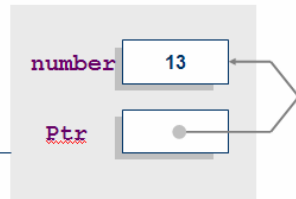


The address operator '&'

- You can find out the address where a particular variable is stored using the & operator followed by the name of the variable.

```
int number = 13;
int *Ptr;
Ptr = &number;
```

sets pointer to the memory location for number



'*' and '&' operators

- What do you expect this program produces and why?

```
#pragma argsused
int main(int argc, char* argv[])
{
    int Value = 12;
    int *Pointer = &Value;

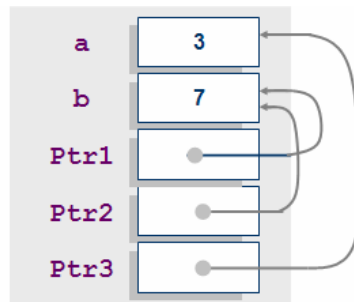
    cout << "Value = " << Value << "\n";
    cout << "Pointer = " << *Pointer << "\n";

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
```

Initialising pointers

- Always remember that pointers MUST be initialised. Otherwise your program will crash! Here are some examples:

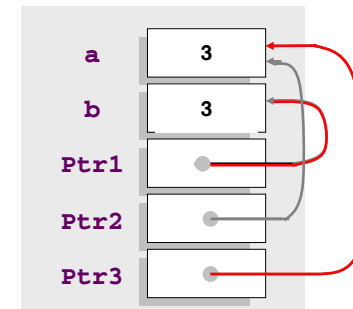
```
int a = 3;
int b = 7;
int *Ptr1 = &b;
int *Ptr2 = Ptr1;
int *Ptr3 = &a;
```



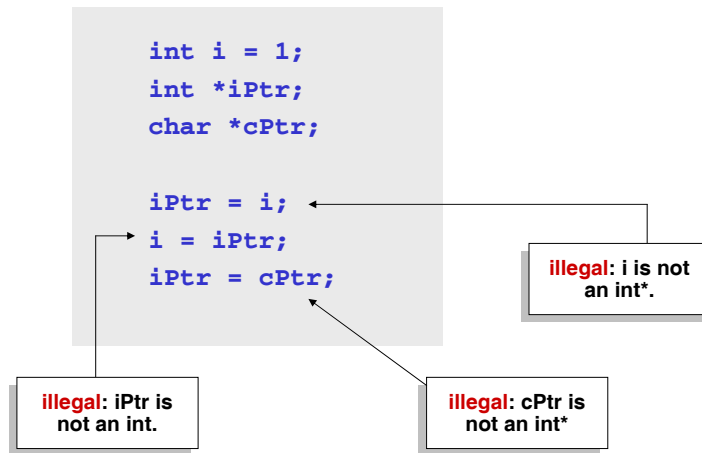
Using pointers

- What would happen if the following code was then also executed?

```
*Ptr1 = *Ptr3;
Ptr2 = Ptr3;
```



Common mistakes



A subtle mistake

- Some people declare pointers with ‘*’ immediately following the type, for example:

```
char* cPtr;
```

- But it can lead to a misunderstanding. For example:

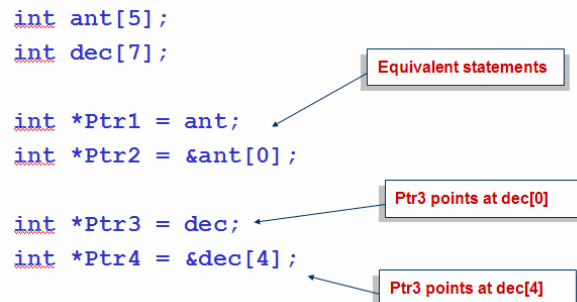
```
char* a, b;
```

really.... char* a;
char b;

- Hence this statement does not lead to the creation of two pointers. Always define only one pointer per statement.

Array and pointer

- In C++, the name of an array is considered to be a **constant pointer**.
- The name of the array is associated with the memory location of the **first element** in the array.
- For example:

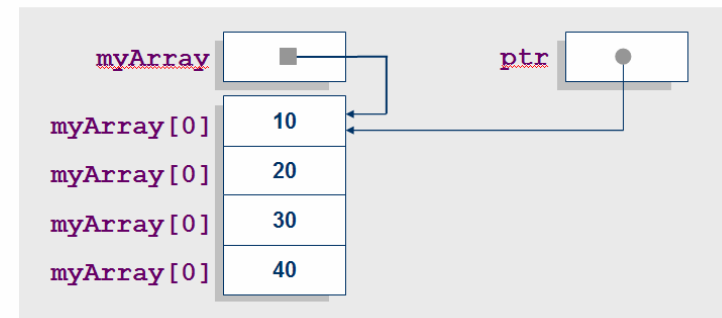


Array and pointer - more....

```

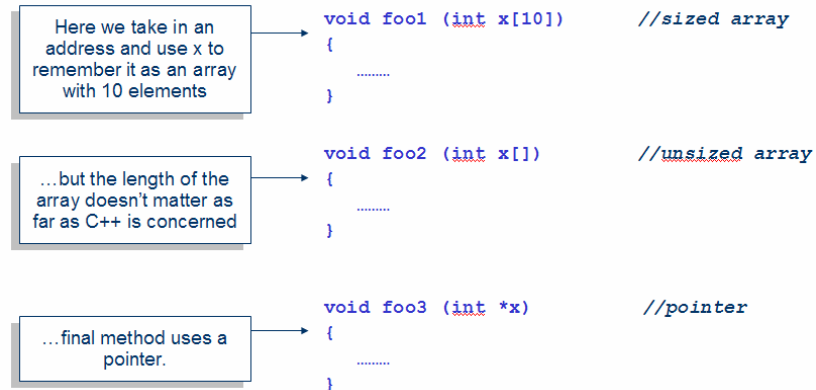
int myArray[4]={10,20,30,40};
int *ptr = myArray;

```



Passing array as parameter into functions

- Array parameters are passed into functions as pointers! (i.e. call-by-reference). These three methods here do the same thing.



Passing Parameters revisited

- In C++, you can pass parameters by reference or by value to a function.
- **Call-by-reference** works almost as if the argument variable is substituted for the formal parameter, not the argument's value
- In reality, the memory location of the argument variable is given to the formal parameter
 - Whatever is done to a formal parameter in the function body, is actually done to the value at the memory location of the argument variable

Call By Reference vs By Value

- Call-by-reference in C++
- Call-by-value

The function call:

`f(age);`

Memory

Name	Location	Contents
age	1001	34
initial	1002	A
hours	1003	23.5
	1004	

`f(age);`

address is passed

value is passed

`void f(int& ref_par);`

`void f(int var_par);`

Call-by-reference equivalent in C

- C++ call-by-reference
- How C does this?

The function call:

`f(age);`

The function call:

`f(&age);`

The function definition:

```
void f(int &ref_par){
    ref_par = ...
}
```

The function definition:

```
void f(int *ref_par){
    *ref_par = ...
}
```

Arguments to main()

- Just as other functions can take arguments, so can your main function.
- Generally you pass information into the main via command line arguments, that follow the programs name when you are executing it.
- There are two special built in arguments : argv and argc, that are used to receive information from the command line.
 - argc – holds the number of arguments entered
 - argv – is an array of **pointers** to these arguments

Example of command line parameter

```
int main (int argc, char *argv[])
{
    if (argc!=2) {
        cout << "you forgot to enter info";
    }

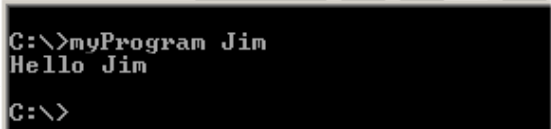
    cout << "Hello " << argv[1];
}

```

check on the number of arguments

statement that outputs the info to the screen.

example of use if we had compiled the code as myProgram.



```
C:\>myProgram Jim
Hello Jim
C:\>

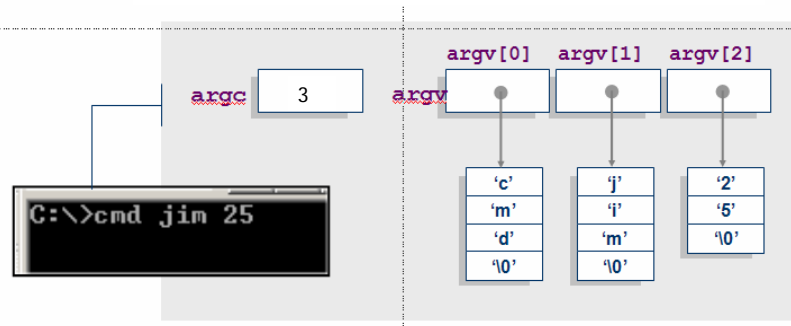
```

Command line argument and pointers

- This is what happens in when command line is used:

```
int main (int argc, char *argv[])
{
    .....
}

```



What to do before the next lecture?

- Complete Exercise C (non-assessed) – See separate sheet
- Either read Lesson 8 & Lesson 15 of the following C++ Tutorial on the web:

<http://www.functionx.com/cppbcb/Lesson08.htm>

<http://www.functionx.com/cppbcb/Lesson15.htm>

Alternatively, read Chapter 6 & 12.1 of Savitch