School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

More about pointers and memory management

Previously on ADDS

- Pointers

```
int *ptr = new int;
*ptr = 6;
ptr = new int;
```

- Stack and Heap
  - Heap fragmentation
  - Memory leak

- Segmentation fault
- Global, Automatic and Dynamic variables

Overview

- Dynamic Array
- Multi-Dimensional Array
- Pointer to functions
- ADT

Dynamic Array

- It is illegal/meaningless to change the pointer value in an array variable.

```
int a[10];
int b[20];
int *ptr;
ptr = b;

a = ptr; // Illegal
```

- For ordinary arrays you must specify the size of the array when you write the program.

- A dynamic array is an array whose size is not specified when you write the program, but is determined while the program is running.

Dynamic Arrays

- Dynamic arrays are created using the *new* operator.

```
double *dArray = new double[array_size];
```

- Dynamic arrays are used like ordinary arrays.

- Remember to call *delete[]* when your program is finished with the dynamic arrays.
- *delete dArray;*
  - Undefined behaviour

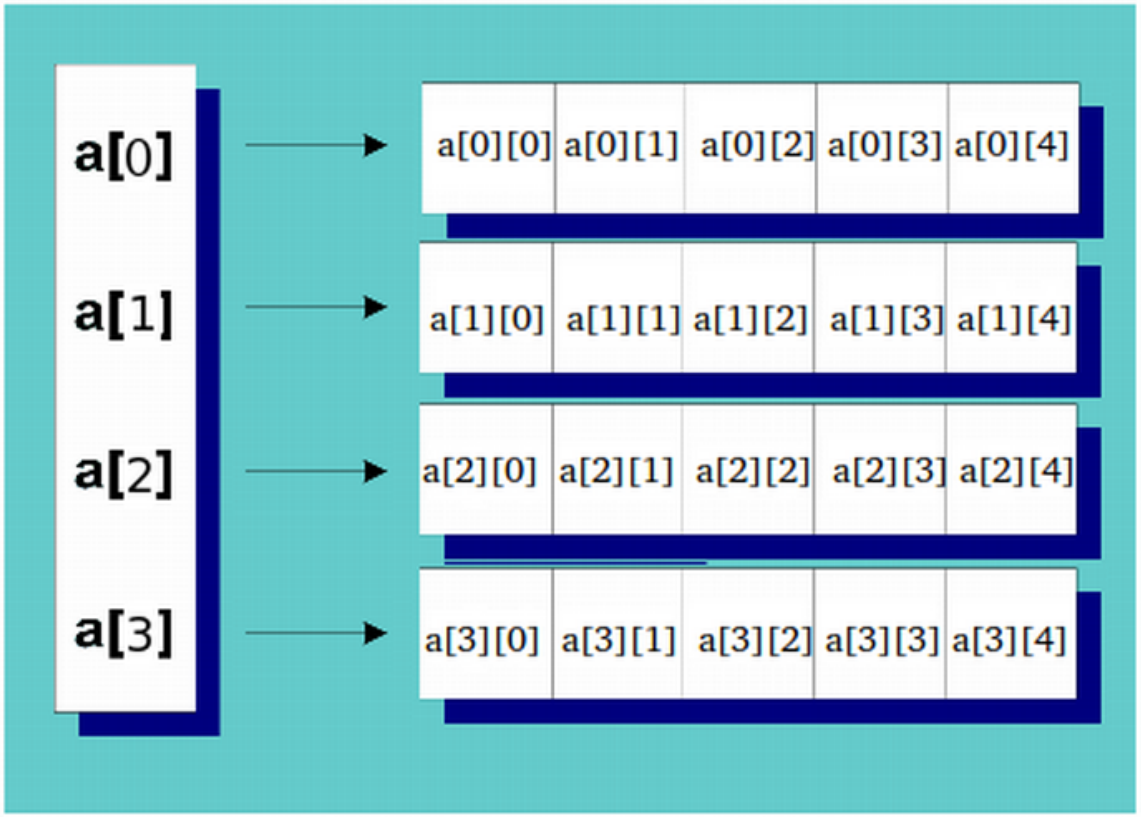
Multi-Dimensional Array

- Two-Dimensional Array

```
dataType arrayName[rowSize][columnSize];
```

- An object of array type contains a contiguously allocated non-empty set of N subobjects of type T.

Multi-Dimensional Array



Multi-Dimensional Array

- Pointer to Pointer (Multiple Indirection)
  - Where is a stored? How about pointers to rows?
  - How about the rows?

```
 1  #include <iostream>
 2  using namespace std;
 3
 4  main() {
 5      int row = 4, col = 5;
 6      int **a;
 7      a = new int*[row];
 8      for (int i = 0; i < row; i++)
 9          a[i] = new int[col];
10  }
11
```

Multi-Dimensional Array

- In C++, you can create n-dimensional arrays for any integer n.

```
int array[15][3][2];
```

Passing Arrays to Functions

- Pass-by-value
- Pass-by-reference
  - Refer the Q1 and Q2 in the worksheet

Returning Arrays From Functions

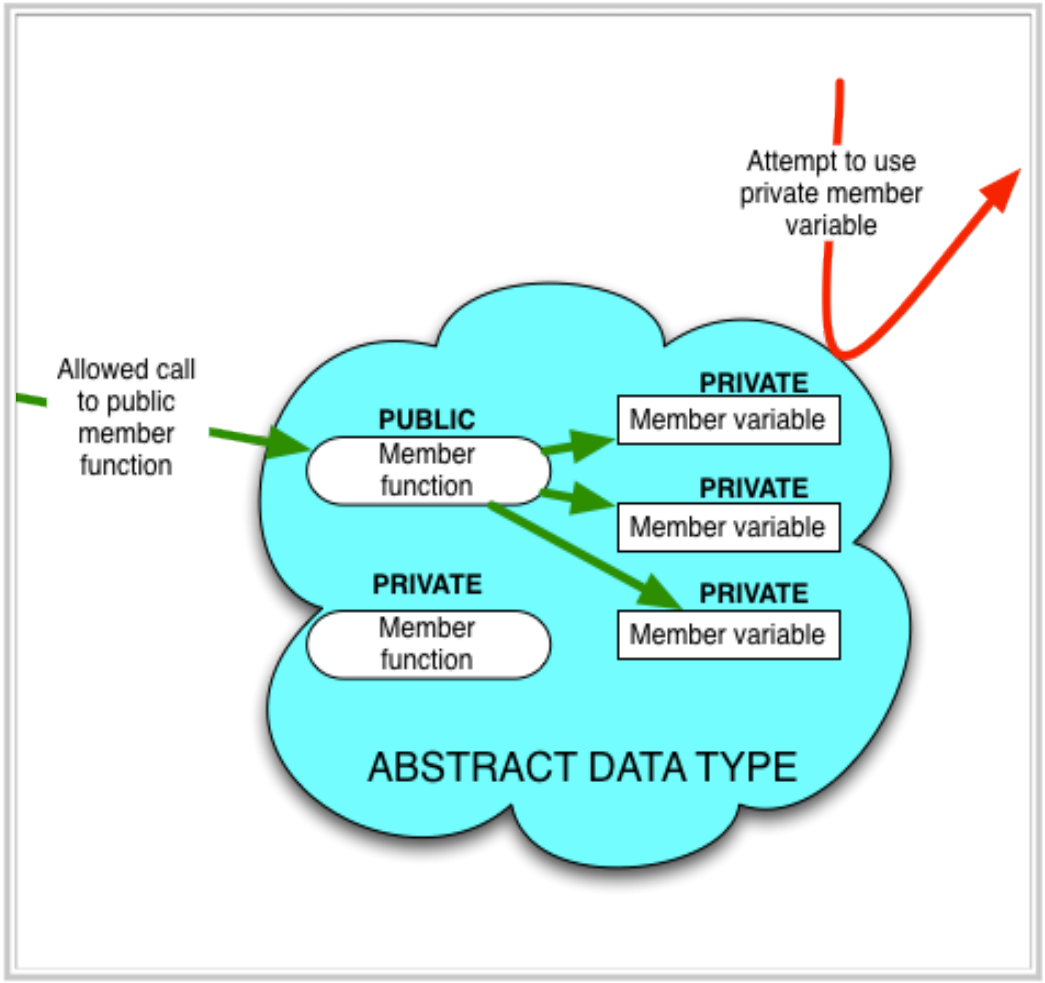
- It is not allowed in C++

- Get around by passing another array argument in the function.
- Return a pointer (Not a good idea to return the address of a local variable)

Pointer to Functions

Refer Q4 in the worksheet

Abstract Data Types



Data Type

- Types are more than just values, they also come with a valid set of operations.
- A data type is the values AND the set of operations defined over these values.

Abstract Data Types

- Suppose we have a type where the public member functions provide a large number of increasingly more complex operations.
- Now, think about that we can do something with the type, but have no idea how it is doing it - or change how it is being done.
- The details have been abstracted away from us.

A data type is called an ADT if the programmers who use the type have no access to the details of the implementation.

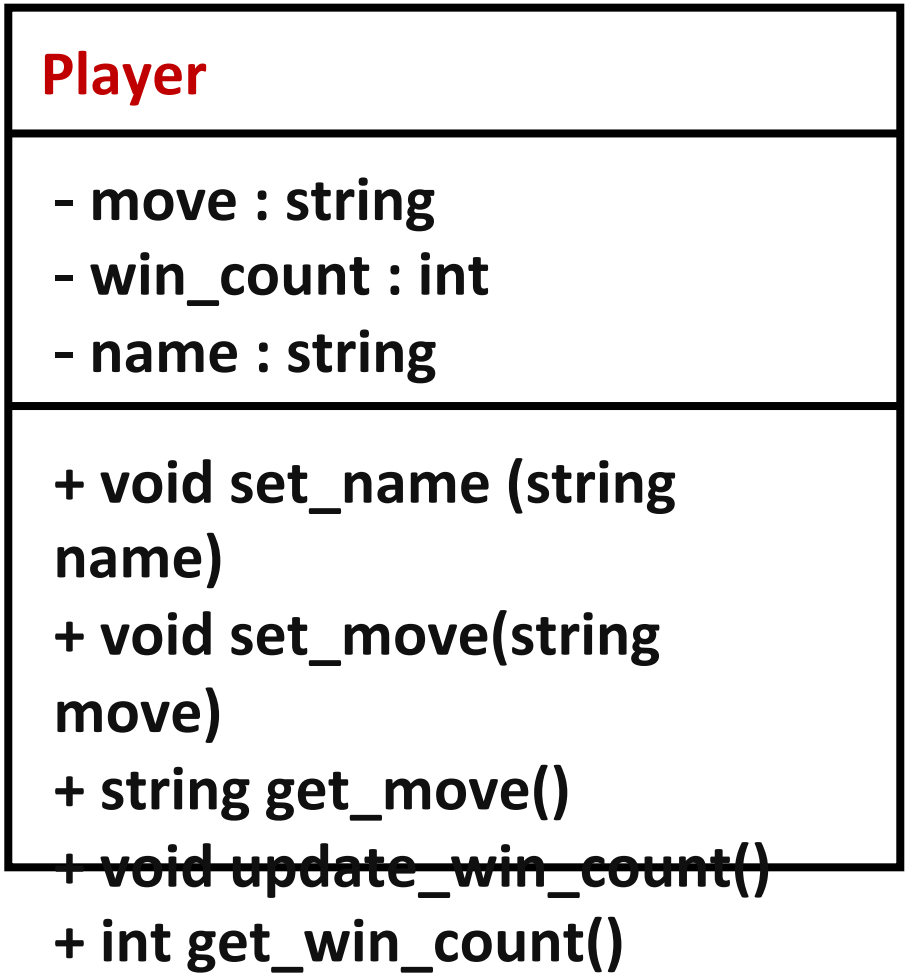
Not all classes are ADTs

- Programmer-defined types are not automatically ADTs.

- Unless defined and used with care, the programmer-defined types can make a program difficult to understand and modify.
- We need to control access to make sure that only part of the behaviour is available to others.
- How do we define the behaviour?

Example

- Recall the definition of class in C++
- How can we create a Player object?
- Do we need to know the implementation of the functions?



Separation

- We need to separate the specification of how the type is used by the users from the details of how the type is implemented.
- Class abstraction is the separation of class implementation from the use of a class

- Rules:
  - Make all member variables private
  - Make the basic operations public and specify how to use them
  - Make any helping functions private

Interfaces

- The set of public member functions in our class, along with a description of what they do, make up the *interface* of the ADT.
- This should be all that someone needs to know to use your ADT.
- At the moment, we're writing the declaration and the implementation in the same file. But we won't always do that.

Implementation

- The implementation of the ADT tells how this interface is realized as C++ code, including:
  - definitions of public functions
  - any private or public variables
  - any private ‘helper’ functions

ADTs and Black Boxes

- From a design point of view, the implementation of an ADT is like a Black Box - you can't see inside it.
- All a programmer can see is your interface.
- A programmer shouldn't NEED to know about the implementation to make the ADT work.
  - Do you know how std::string or '+' are implemented?
- This is also known as *information hiding*.


THE UNIVERSITY
of ADELAIDE