UNIVERSITY OF LIVERPOOL

Application Development with C++ (ELEC362)

Lecture 5: Pointers and Functions

mihasan@liverpool.ac.uk

Previous lecture

- Three types of control structures (sequential, selection, and repetition structures) were discussed.
- If-else and switch-case structures were discussed under selection structures.
- For loop, While and Do-while loops were discussed under repetition structures.
- The concept of event loops was discussed.

This lecture

- What is covered in this lecture?

  1. Pointers. 2. Functions. 3. Exception handling.

- Why it is covered?

  1. Pointers are vital for efficient memory management of any programme.

  2. Functions are the building blocks of classes.

  3. Exception handling is the professional way to deal with errors in a programme.

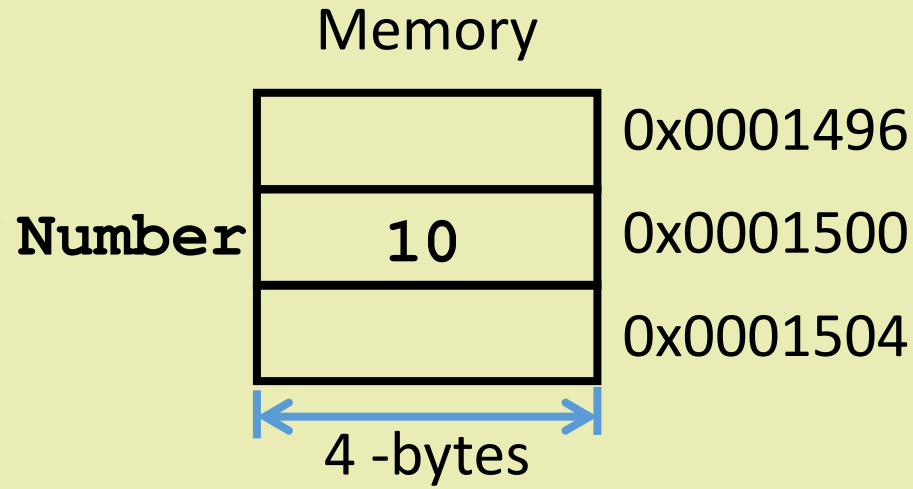
- How are topics covered in this lecture:

  3 source codes

Pointers definition

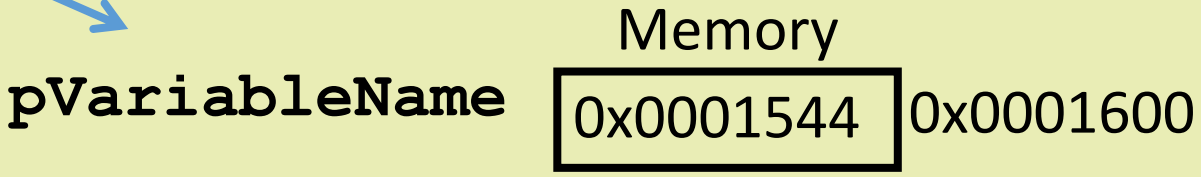
- Pointers are variables that store the memory address of a variable of the same type , or memory location.

```
int Number = 10;
```



- Syntax:

```
DataType *pVariableName;
DataType* pVariableName;
```



- Google style code requires the dereferencing operator “*” to be attached to either the data type or the variable’s name.

Pointers naming and initialisation

- It is recommended to start the name of a pointer by “p”.
- To initialise a pointer, the referencing operator “&” should be used.
- Examples:

```
int Number{99};
int *pNumber;//Declare pointer
pNumber = &Number;//Initialise it
```

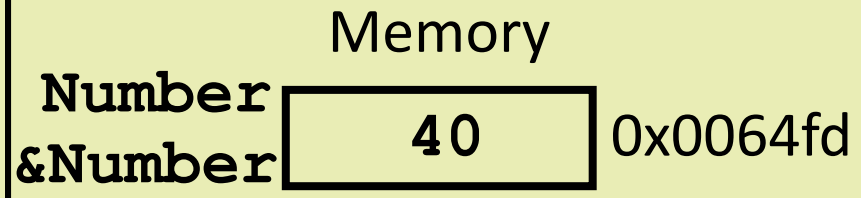
```
int Number{99};
//Declaration & initialization
int *pNumber{&Number};
```

- The datatype of the pointer has to match the datatype of the variable it is pointing to.

Referencing and Dereferencing operators

- The reference operator “&” gives the address of a given variable in the memory.
- It is possible to define another name of the same memory address using “&”.
- The content of a memory address is accessible using the dereferencing “*” operator.

```
int Number = 40;
int *pNumber = &Number; // Say its 0x0064fd
int &rNumber = Number; // “Alias” variable
std::cout << Number << endl;
std::cout << &Number << endl;
std::cout << rNumber << endl;
std::cout << pNumber << endl;
std::cout << *pNumber << endl;
```

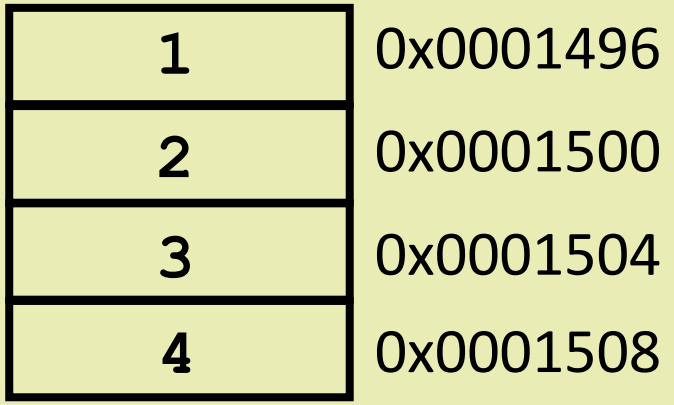


- Pointers can be modified while references cannot.

Pointers and arrays

- The name of any array in C++ is a memory address, which can be used to initialize a pointer to access and manipulate the elements of the array.
- Example:

```
int Data[4]{1,2,3,4};
int *pData{nullptr}; // Points nowhere
pData = Data; // Equivalent to pData=&Data[0];
std::cout<<*(pData+2)<<endl;
std::cout<<Data<<endl;
```



Practical note:

- Make sure your code does not access memory locations beyond and array.
- After a pointer is not needed, reinitialise it to **`nullptr`**

Advantages of Pointers

- Programs use static memory (the stack) and dynamic memory (the heap).
- Pointers are the only way the programme can have access to dynamic memory.

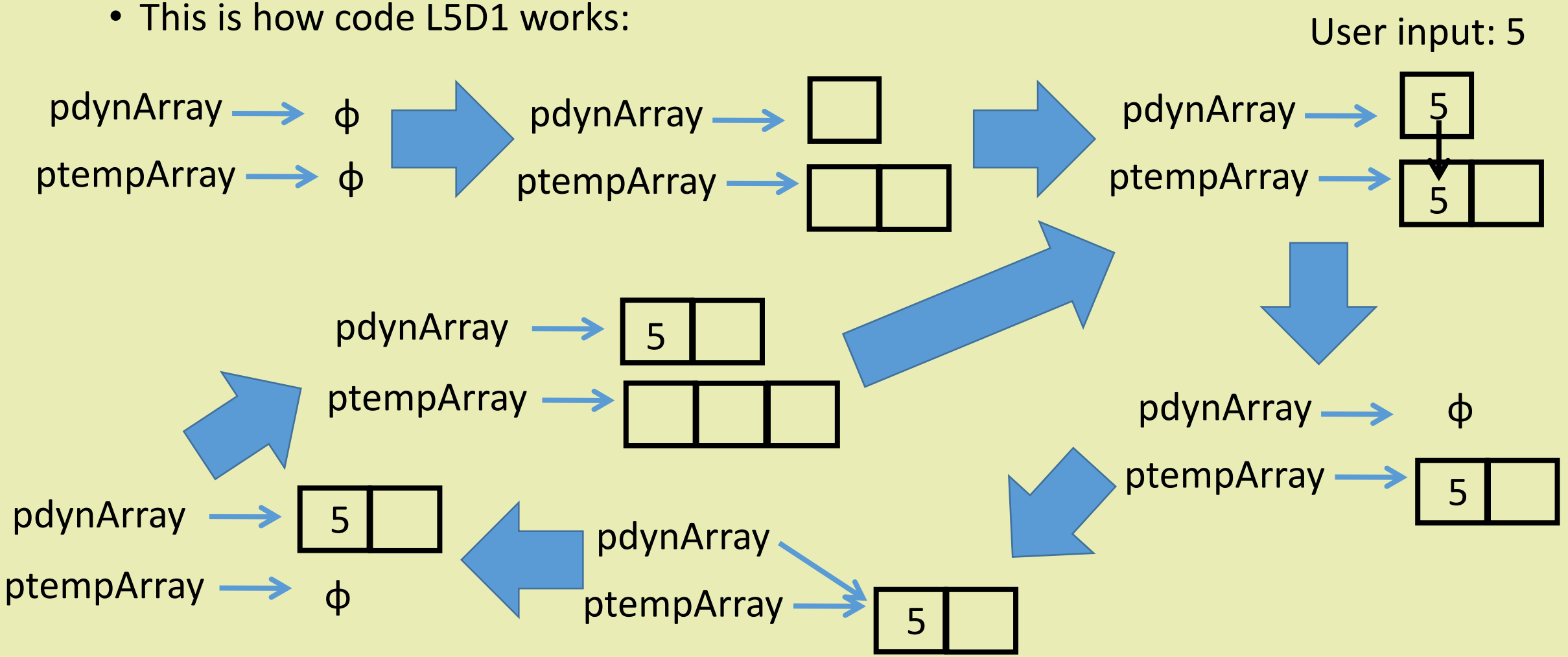
Aspect	Static memory	Dynamic memory
Allocation	Allocation is done at compile-time only	Allocation can be done at compile time or run-time
Lifetime	As long as the programme is running	Can be controlled
Size	Fixed	Variable

- Pointers can access dynamic memory through `new` keyword (for allocation) and `delete` (for freeing memory). `new` and `delete` (`delete []` for arrays) must operate on the same memory location.
- Go to L5D1.cpp

Practical note: Maximise the use of dynamic memory in your code.

Dynamic memory allocation algorithm

- This is how code L5D1 works:



Functions in C++

- A function is a group of statements that together perform a defined task.
- Syntax:

```
return_type FunctionName(datatype argument1, …,…)
            { // Statements;
              return output; }
```

- Naming convention follows Google C++ style code (every word starts with capital).
- Few rules: 1. Functions has to be declared before main function.

  2. Every function only returns one value.

  3. The datatype of the output has to match the return type of the function.

  4. Return type of `void` is used when the function has no return.

Functions declarations and arguments passing

- There are two styles of declaring a function (with or without function declaration):

```
int MyFun(int argument); //declaration

int main () {
              //Statements;
            }
int MyFun(int argument){
                        //Statements;
                       }
```

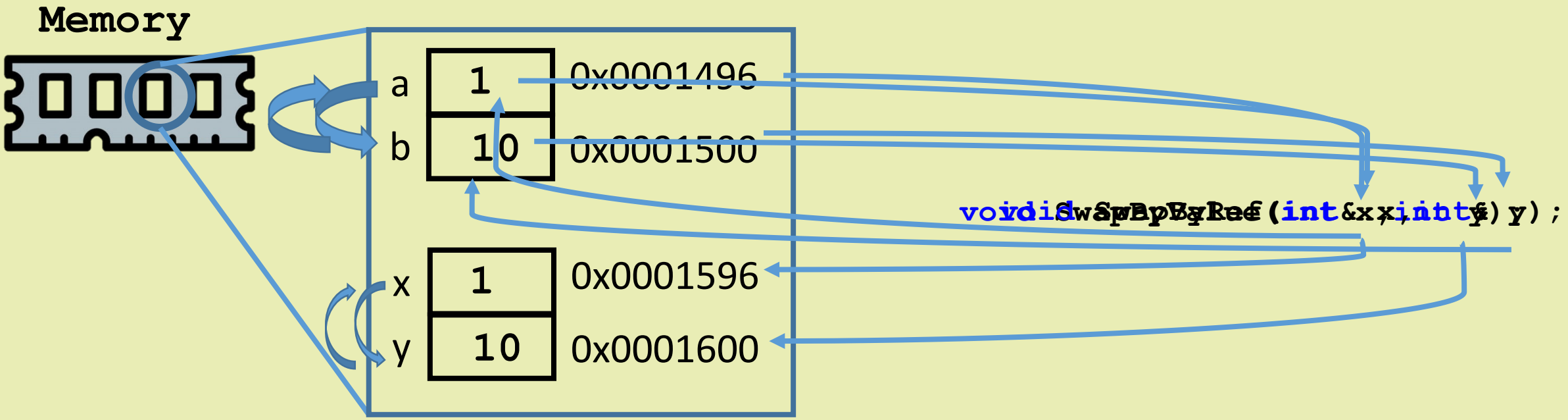
```
int MyFun(int argument){
                        //Statements;
                       }

int main () {
              //Statements;
            }
```

- It is more common in codes to follow function declaration style.
- The arguments of functions can be passed by value, by reference or using pointers.
- Go to L5D2.cpp

Arguments passing in functions

- This is how code L5D2 works:



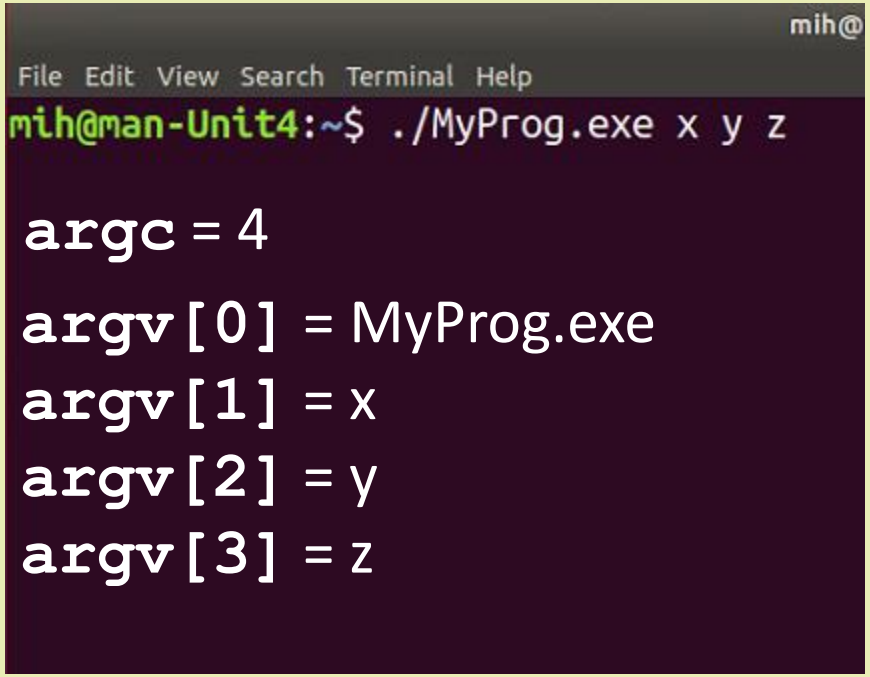
Practical note: Pass variables by reference or pointers whenever possible.

Arguments of the `main` function

- In Linux environment, it is more common to run programmes in batch mode (running a programme simply by calling it from a command line).
- In such context, the main function can have parameters, most commonly:

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

- **`argc`** is a variable containing the number of arguments passed to the programme through command line. **`argv[]`** is an array of pointers pointing to the arguments passed through the command line.



Function overloading

- Function overloading is defining multiple functions with the same name but different parameters. The compiler can decided which one to call based on the parameters passed at calling.
- Example:

```
// Two integer version
int Max(int x,int y); {return (x > y ? x : y);}
// Three integer version
int Max(int x,int y,int z); {int m = (x > y ? x : y);
                             return (z > m ? z : m);}
//Two double version
double Max(double x,double y); {return (x > y ? x : y);}

void main () {  int c = 2, d = 4;
                double g = 2.3 , h = 5.2;
                Max(g,h);
                Max (c,d);
             }
```

Function templates

- Function templates are efficient way to generate overloaded functions for different data types.
- The idea is to pass the data type as it is a parameter to the function.

```
template <typename T> // T is the arbitrary datatype
T Max(T x,T y){return (x > y ? x : y);}

void main () { int c = 2, d = 4;
               double g = 2.3 , h = 5.2;
               char j='y', k='w';
               Max<double>(g,h); // T is double
               Max<int>(c,d);   // T is integer
               Max<char>(j,k);  // T is character
             }
```

Exceptions and error handling

- Exceptions are run-time anomalies or errors a program encounters during execution.
- When developing a software, the developer should anticipate the possible run-time error and handle them through exception.
- Exception handling can be done using three keywords:

1. `try`: Defines the code block where the exception may occur (be thrown).
2. `throw`: Used to declares the exception.
3. `catch`: Defines the response of the program if a particular exception occurs.

- Go to L5D3.cpp

Practical note: Handle errors using try-throw-catch structure in your code.

Summary

- The definition of pointers, their use and their advantages were discussed.
- Differences between two types of memory were discussed.
- Functions definitions and declarations were discussed.
- Functions overloading and templates were discussed .
- Exceptions and error handling were discussed.