Programming with C++

LSEG Technology

04th November 2022

Agenda

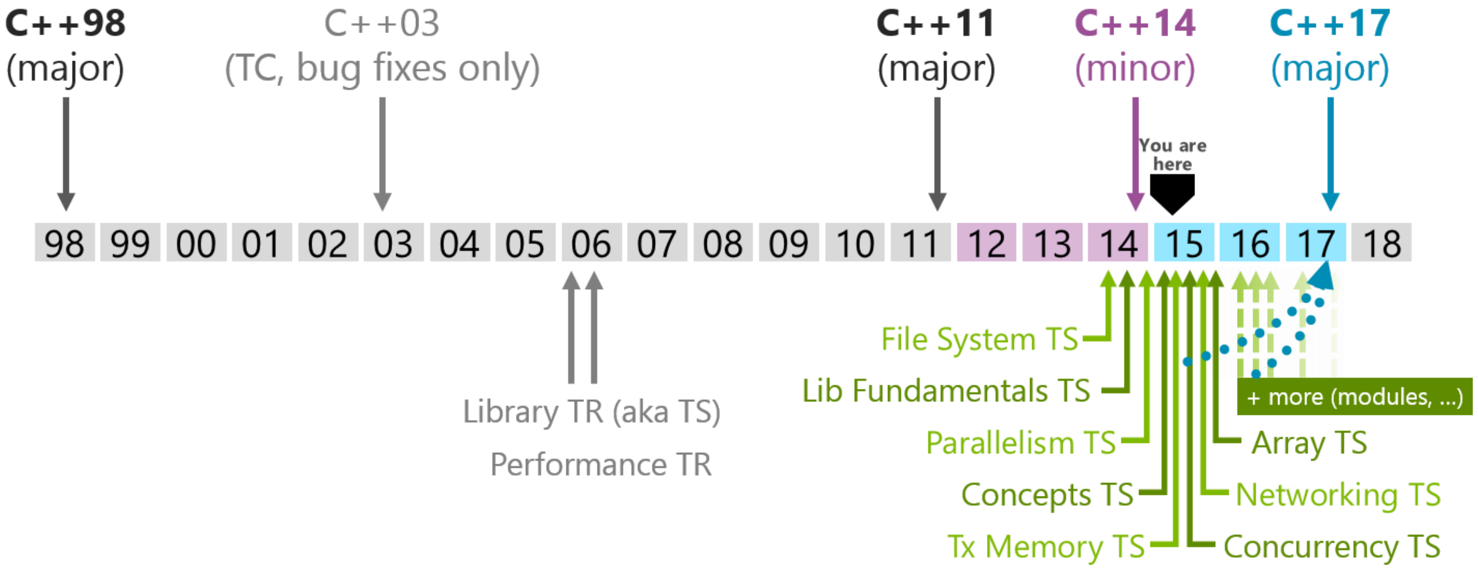
- Why C++
- History of C++
- The First Program
- The Tools
- Stages of Compilation
- Data Types and Operators
- Control Structures
- Functions
- Pointers

Why C++

- Nothing that can handle complexity runs as fast as C++
- In embedded areas, image processing, some telecom applications and some financial applications etc. **C++ rules**.
- 3rd most popular language after Python and C (IEEE Spectrum)

History of C++

- 1980 : C with Classes by Bjarne Stroustrup.
- 1983 : C with Classes redesigned and called C++
- 1989 : C++ 2.0
- 1998 : The first official ISO standard (C++98)



***Source*: http://herbsutter.com/2013/10/03/trip-report-fall-iso-c-standards-meeting/**

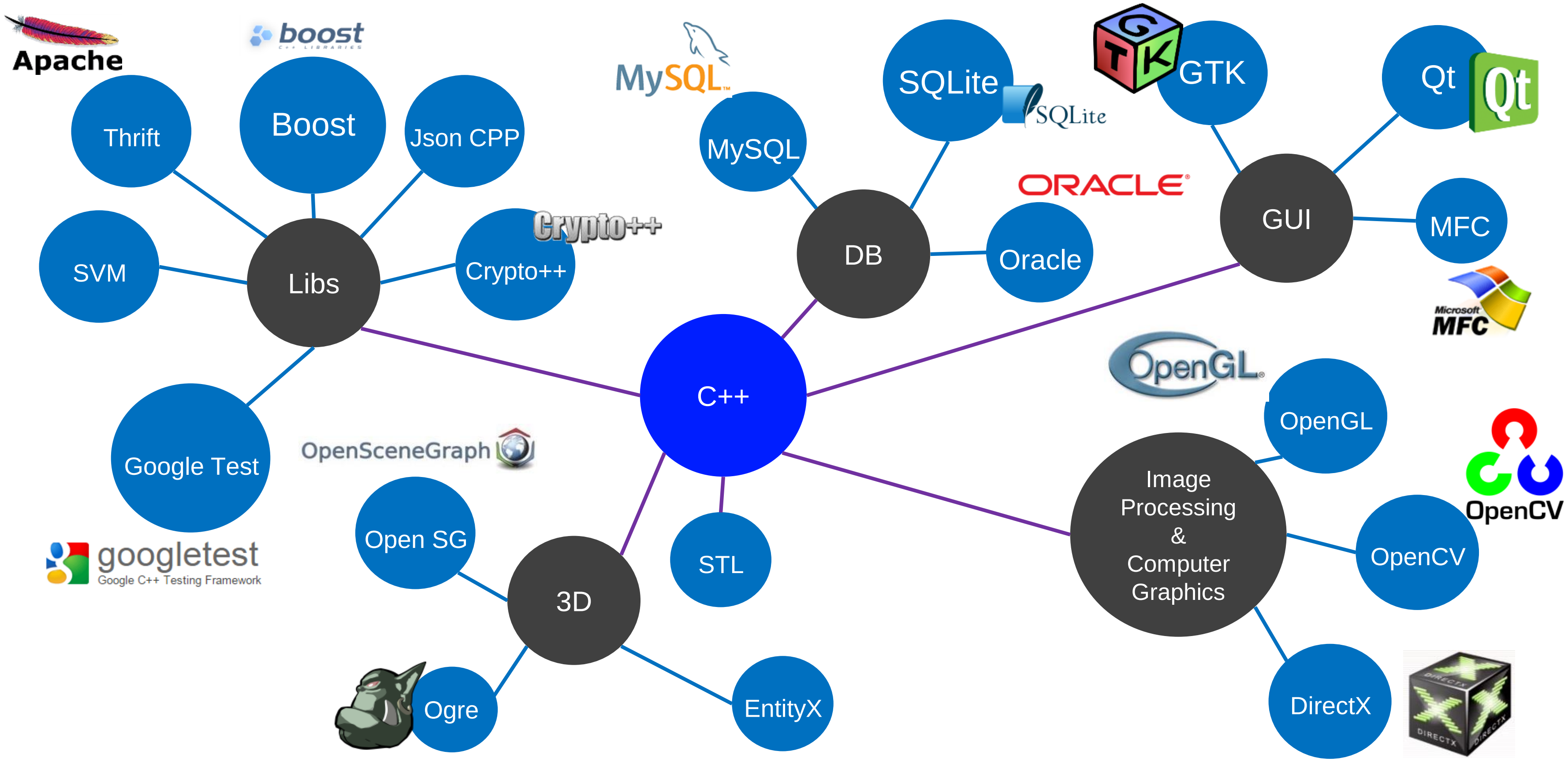
Hello World!

```
#include <iostream>

int main(int argc, char* argv[])
{
  std::cout << “Hello World!\n”;
  return 0;
}
```

What are these parameters?

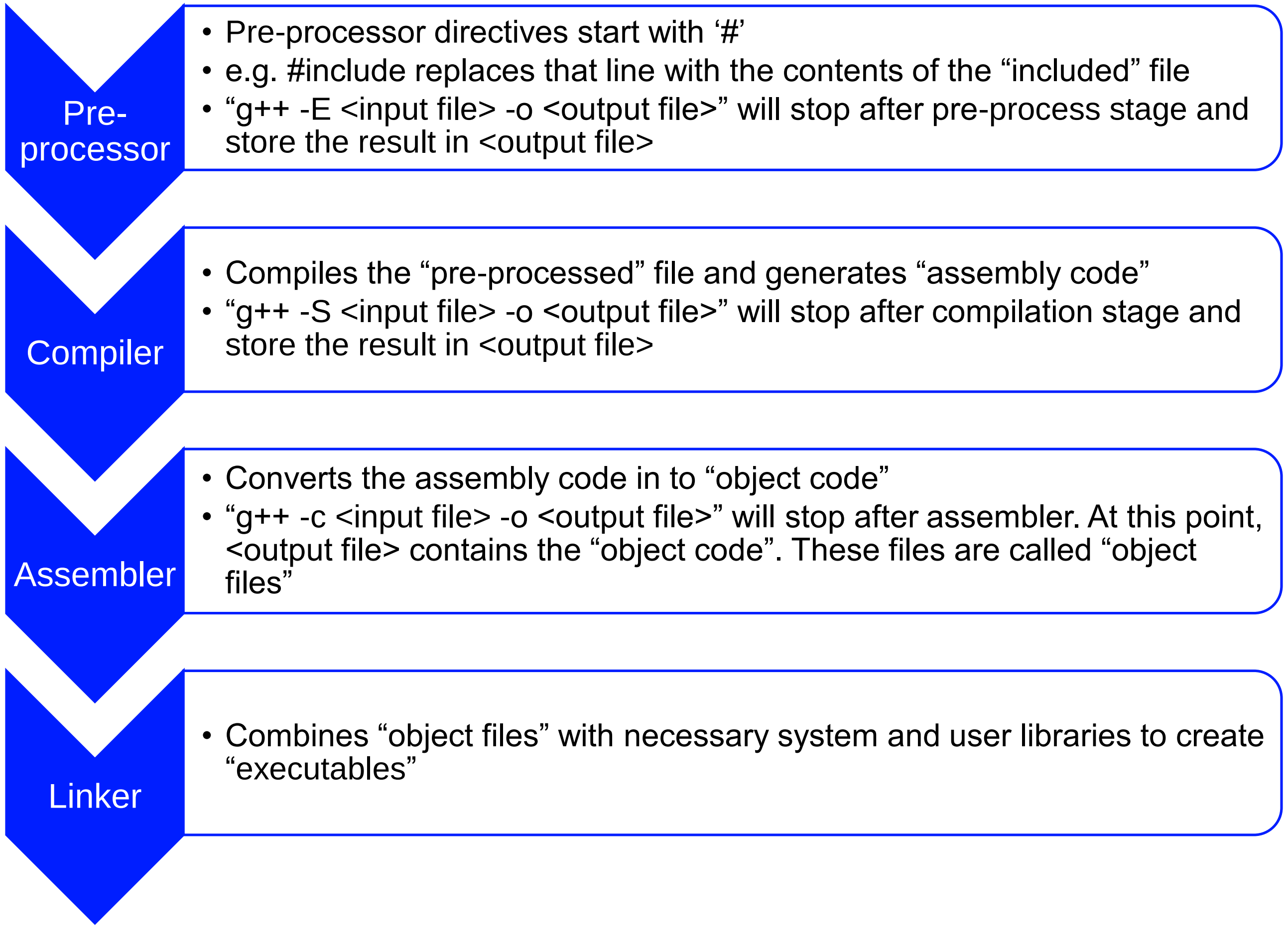
Why return a 0?
Main function in C++ has an integer return type.
0 traditionally indicates that the program ran successfully.

Library Support for C++



LSEG

Stages of Compilation



Pre-processor

- Pre-processor directives start with '#'
- e.g. #include replaces that line with the contents of the "included" file
- "g++ -E <input file> -o <output file>" will stop after pre-process stage and store the result in <output file>

Compiler

- Compiles the "pre-processed" file and generates "assembly code"
- "g++ -S <input file> -o <output file>" will stop after compilation stage and store the result in <output file>

Assembler

- Converts the assembly code in to "object code"
- "g++ -c <input file> -o <output file>" will stop after assembler. At this point, <output file> contains the "object code". These files are called "object files"

Linker

- Combines "object files" with necessary system and user libraries to create "executables"

Data Types

Category	Types	Meaning	Example	Notes
boolean	bool	true or false	true	
character	char, wchar_t, char16_t, char32_t	a single ASCII character	‘c’	char16_t, char32_t are C++11 only
floating point	float, double, long double	a number with a decimal	3.14159	
integer	short, int, long, long long	a whole number	64	long long is C99/C++11 only
void	no type	void	n/a	

Variables

- Variables are names to pieces of memory
- int x;   - declare (and define) an integer variable
- int x = 5;   - declare a variable and initialize to 5

Operators

Category	Operator(s)	Example	Notes

Assignment	=	x = 5;
		x = y = 10;	

Arithmetic	+, -, *, /, %	x = 11 % 3	x gets value ‘2’
Compound assignment	+=, -=, *=, /=	x += 5;	Same as x = x + 5;
Increment and decrement	++, --	x++ --x	Has suffix and prefix forms
Comparison	==, !=, <, >, <=, >=		
Logical	!, &&, ||	if ((a > 5) && (b < 10))	
Bitwise	&, |, ^, ~, <<, >>		

Control Flow

- Three logical constructs
  - Sequence, Selection, Repetition
- Selection
  - if, if…else, if… else if… else
  - switch
- Repetition
  - for (int i = 0; i < n; i++) { … }
  - while (true) { … }
  - do { … } while (true)

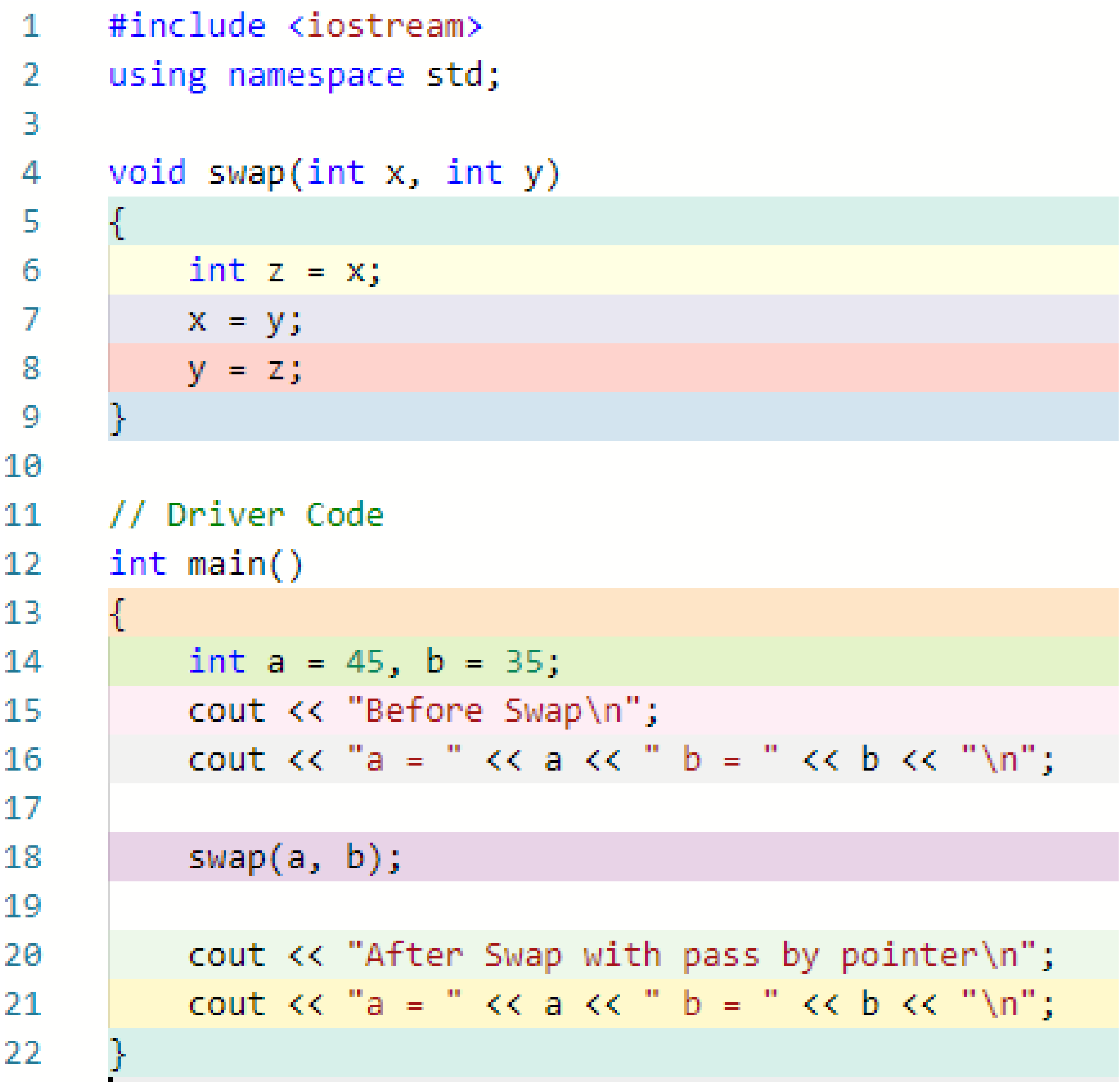
Arrays

- An array is a collection of variables of the same type
- It’s a convenient way to access multiple variables with a single name and an index value. Array indexes are zero-based (i.e. start at zero)
- int a[10]; - creates an un-initialized array of 10 integers
- a[0] = 5; - stores 5 in the first element of array. a[9] is the last element
- std::cout << a[4] – prints the 5th element in array to console
- Array can be multi-dimensional
- int a[10][10] - define a 10 x 10, 2D array (matrix) of integers

Exercise

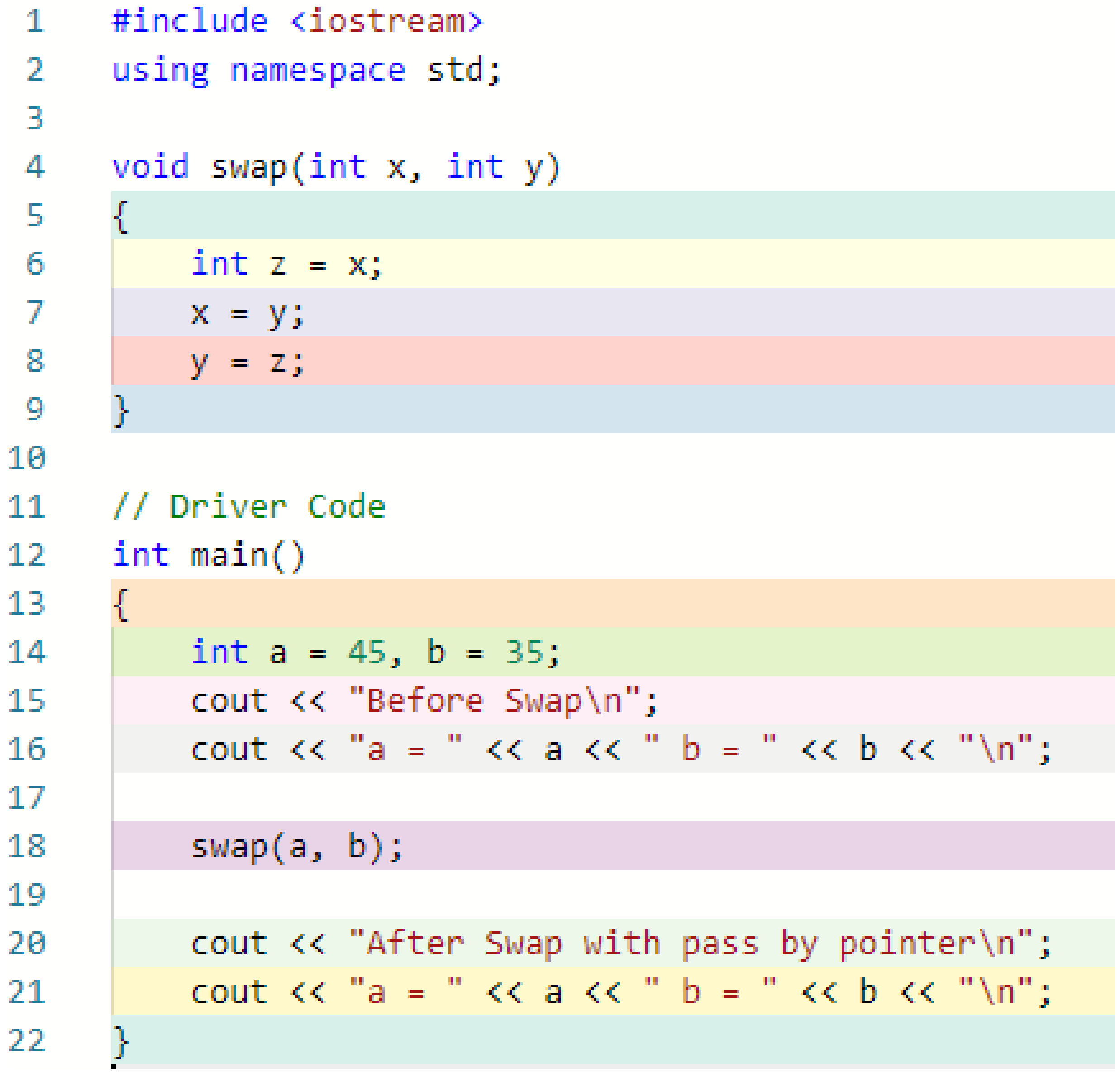
- Write a program to generate the multiplication tables up to 12 x 12.

Exercise : What is the output of below program ?

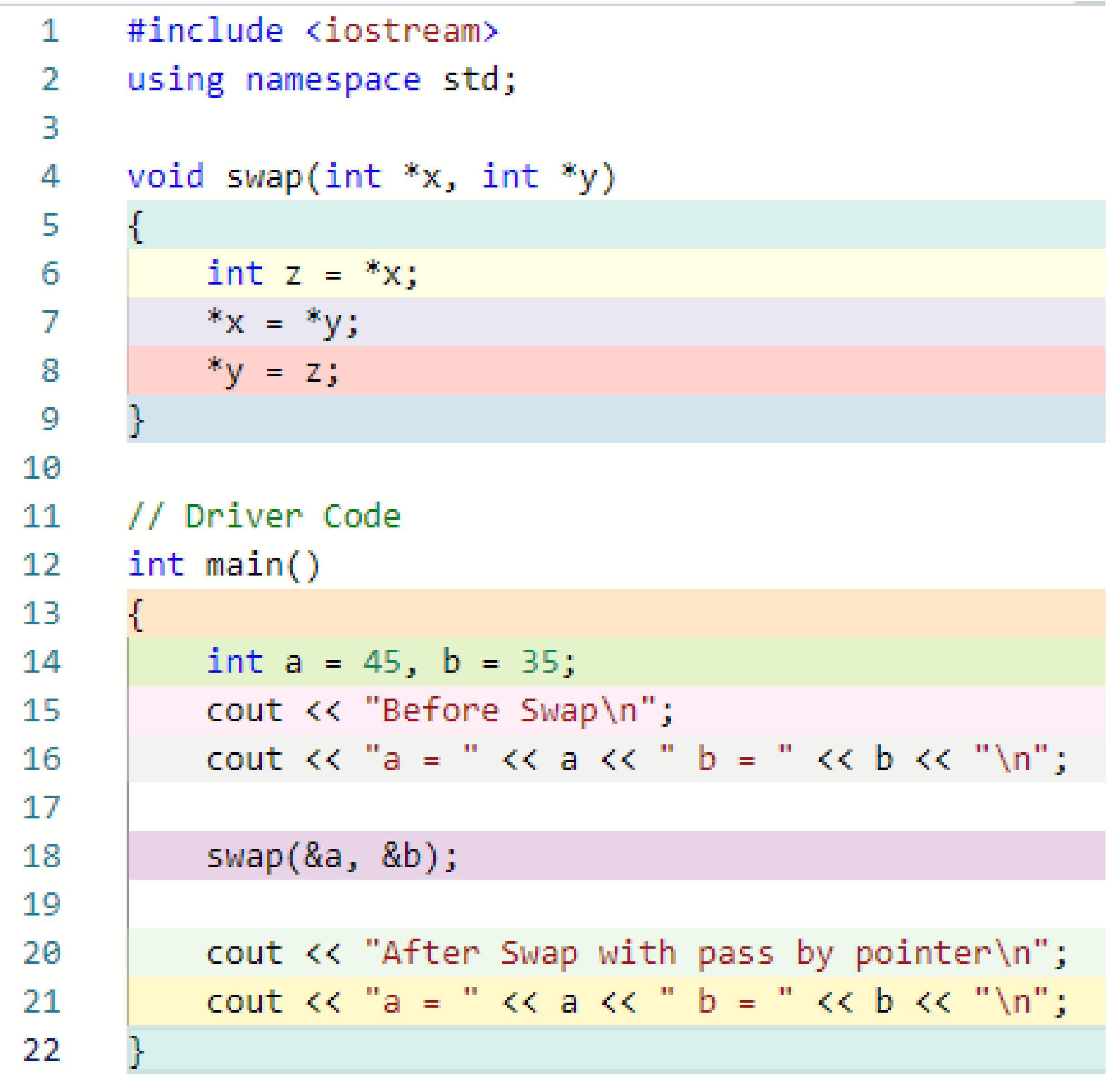


```cpp
#include <iostream>
using namespace std;

void swap(int x, int y)
{
    int z = x;
    x = y;
    y = z;
}

// Driver Code
int main()
{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << " b = " << b << "\n";

    swap(a, b);

    cout << "After Swap with pass by pointer\n";
    cout << "a = " << a << " b = " << b << "\n";
}
```

Exercise 2 : What is the output of the second program ?



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

void swap(int x, int y)
{
    int z = x;
    x = y;
    y = z;
}

// Driver Code
int main()
{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << " b = " << b << "\n";

    swap(a, b);

    cout << "After Swap with pass by pointer\n";
    cout << "a = " << a << " b = " << b << "\n";
}
```



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

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

// Driver Code
int main()
{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << " b = " << b << "\n";

    swap(&a, &b);

    cout << "After Swap with pass by pointer\n";
    cout << "a = " << a << " b = " << b << "\n";
}
```

Functions

- Function signature
- Function declaration vs. definition
- Function parameters
- Pass by value, pass by reference
- Recursion

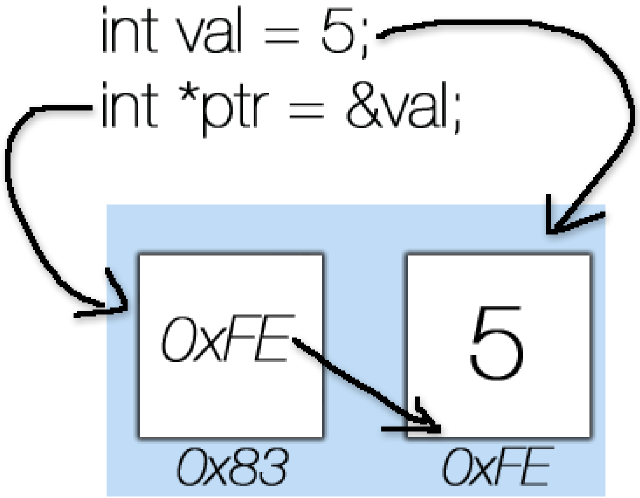
Exercise

- Write a program to calculate the factorial of a given integer number

Pointers and Dynamic Memory

We will be talking about raw pointers here.. Smart pointers will be in a next session

- A pointer is a variable that holds a memory address
- Forget int*, char* or A*. It will be storing an integer value that points to a memory address.
- Type of Pointer: says the data at this address is said to be of this type.
- We can write code without pointers. But, this is im
- Free store (heap)
- Memory allocation and de-allocation
- **new** and **delete** operators
- Address-of and contents-of operators


```
int val = 5;
int *ptr = &val;
```

References

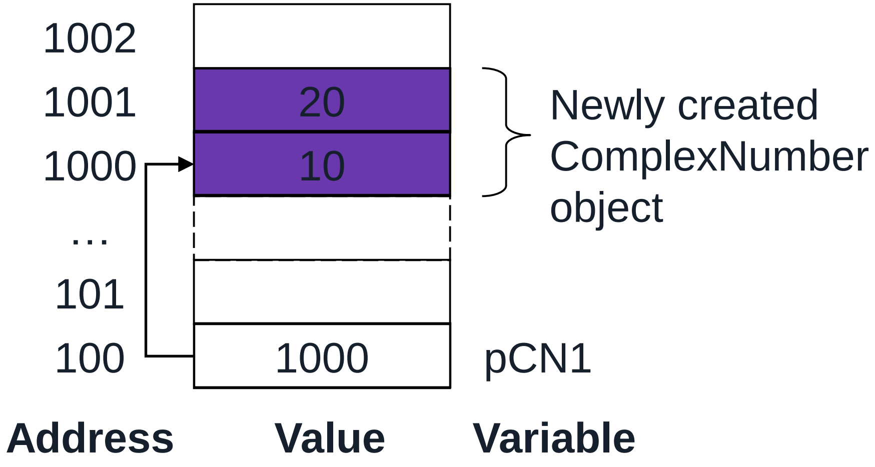
- Not a variable that actually exists
- Just an alias
- Cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.
- Once a reference is initialized to an object, it cannot be changed to refer to another object.
- A reference must be initialized when it is created.

Memory Allocation

Mechanism	Location	Create method	Create time	Destroy method	Destroy time
Locally	on the stack	Declare variable in a block / function scope	On demand	Automatic	When go out of the scope of the variable’s block / function
Statically	on the static memory area	static keyword / global variables	When code is linked to the running program	Automatic	When exit() calls or when program terminates
Dynamically	Heap (free store)	new, new[]	On demand	delete, delete[]	On demand

Creating class objects in free store

```
ComplexNumber *pCN1
        = new ComplexNumber(10, 20);
```



pCN1 Points to the start of object

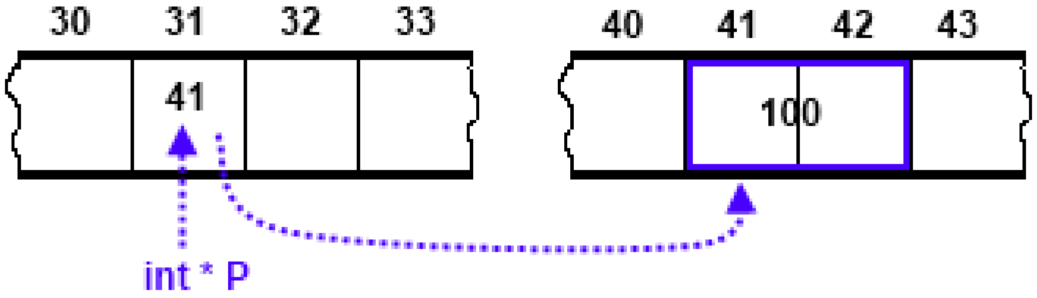
pCN2 Points to the start of object array

C++ Pointers

- Pointers are variables that store memory addresses
- Pointers refer to a memory address that stores a specific type of data
- Data type of a pointer is denoted using " *data_type_of_var** " syntax
- Reference/Dereference operators

  - Consider int A and ClosedShape* S;
  - Then

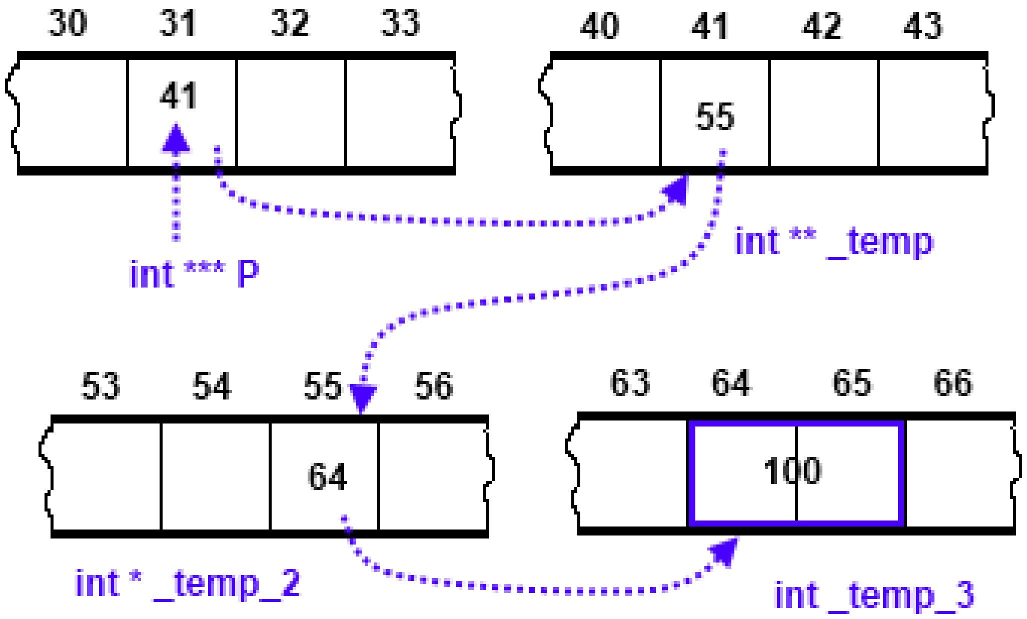
```
Int * P = &A;
int B = *P;
float P = (*S).getArea();
float Q = S->getArea();
```



Pointer to Pointer

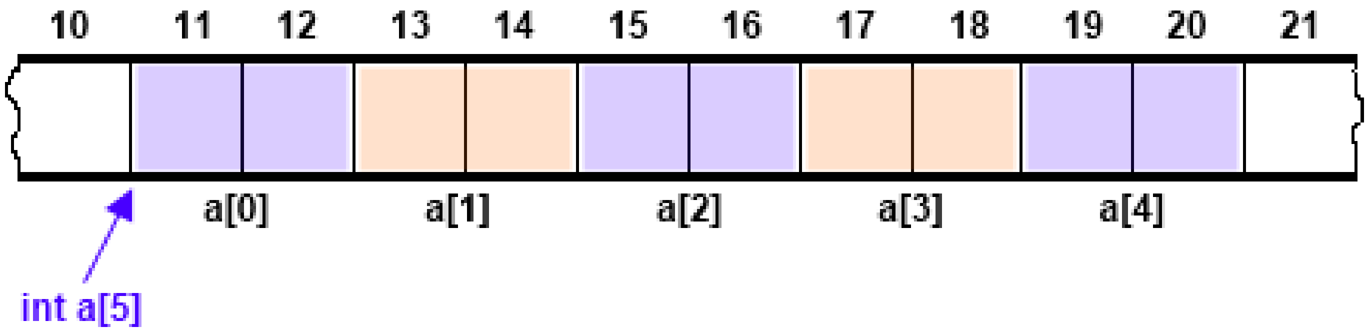
- Pointers can point to other pointers and so on...
  - e.g. ((int *)*)* P

- Dereferencing
  - Consider int*** P
  - *P is of “ int** “ type
  - *(*P) is of “ int * “ type
  - *(*(*P)) is of “ int “ type



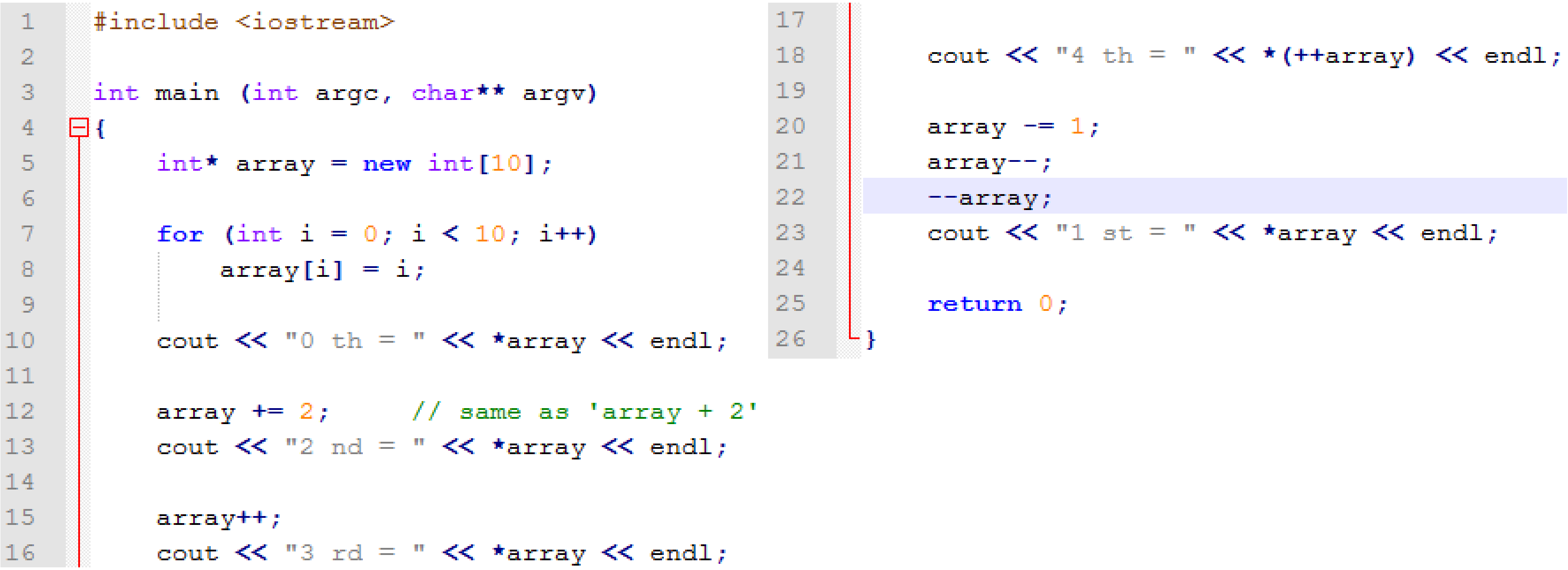
Pointers and arrays

- C++ array variables use consecutive memory block without padding to store data
- Array variable represents the starting address of the memory block
  - Consider the array “ int a[5] “
  - Both “ a “ and “ &a ” represents the starting address of the memory block



Pointer arithmetic

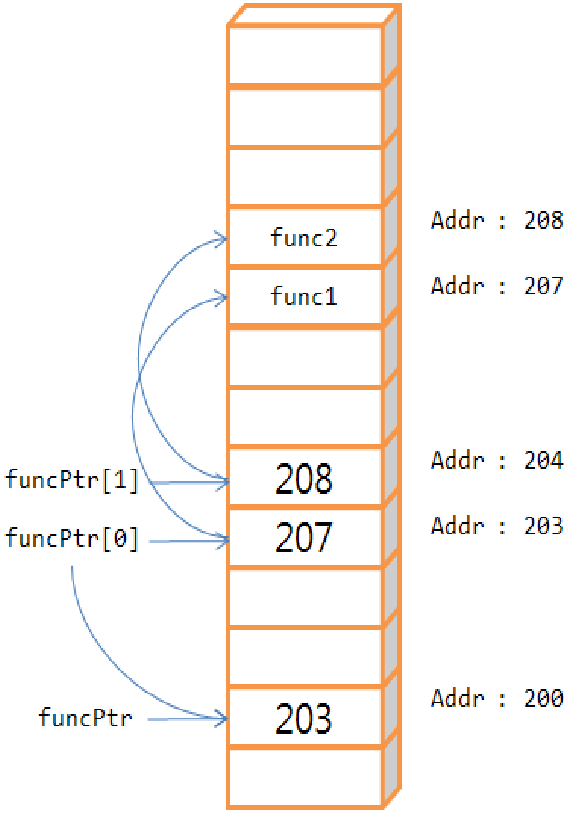
- Pointers support addition and subtraction operators
  - e.g.



```cpp
#include <iostream>

int main (int argc, char** argv)
{
    int* array = new int[10];

    for (int i = 0; i < 10; i++)
        array[i] = i;

    cout << "0 th = " << *array << endl;

    array += 2;      // same as 'array + 2'
    cout << "2 nd = " << *array << endl;

    array++;
    cout << "3 rd = " << *array << endl;

    cout << "4 th = " << *(++array) << endl;

    array -= 1;
    array--;
    --array;
    cout << "1 st = " << *array << endl;

    return 0;
}
```

Function pointers

- C++ allows defining variables that points to functions
- This allows a function to be passed as a argument to another function.
- Function pointers are used internally when handling virtual methods in c++ (i.e. this->__vptr member)
- How ?
- Syntax
  - return_t (*[ptr_name]) (arg1_t, arg2_t);



References

- LearnCpp.com
- CPlusPlus.com Tutorial
- Tutorialspoint.com
- The Cherno