Introduction to C++

Today’ Class

- **Agenda**
  - What is C++ & Why ?
  - History & Standards
  - Fundamentals
    - Data types
    - Input & Output
    - Function
      - Overloading
      - Inline
    - Array

Era of C

C Language

- Developed by Dennis Ritchie
  - Started in 1972, first published in 1978 and approved in 1983
  - For developing operating systems
    - Unix OS
- Flexible and Concise
- Extremely powerful
  - Works efficiently for low-level interactions
    - Ex. OS, drivers, memory, etc.
- High performance
- Adopted by the ISO as ISO/IEC 9899:1990 aka C90 in 1990
  - C99 in 1999 and C11 in 2011

Era of C

- **Problems**
  - Very low-level language
  - No runtime type checking
  - Missing OO concepts
  - Security & Design Issues

Era of C++

- **C++**
  - General purpose programming language
  - Support OO principles and Generic Programming
  - Based on C
    - Borrows best concepts from C
  - Also called as “C with Classes” or “Extended C”

C++ vs. Java

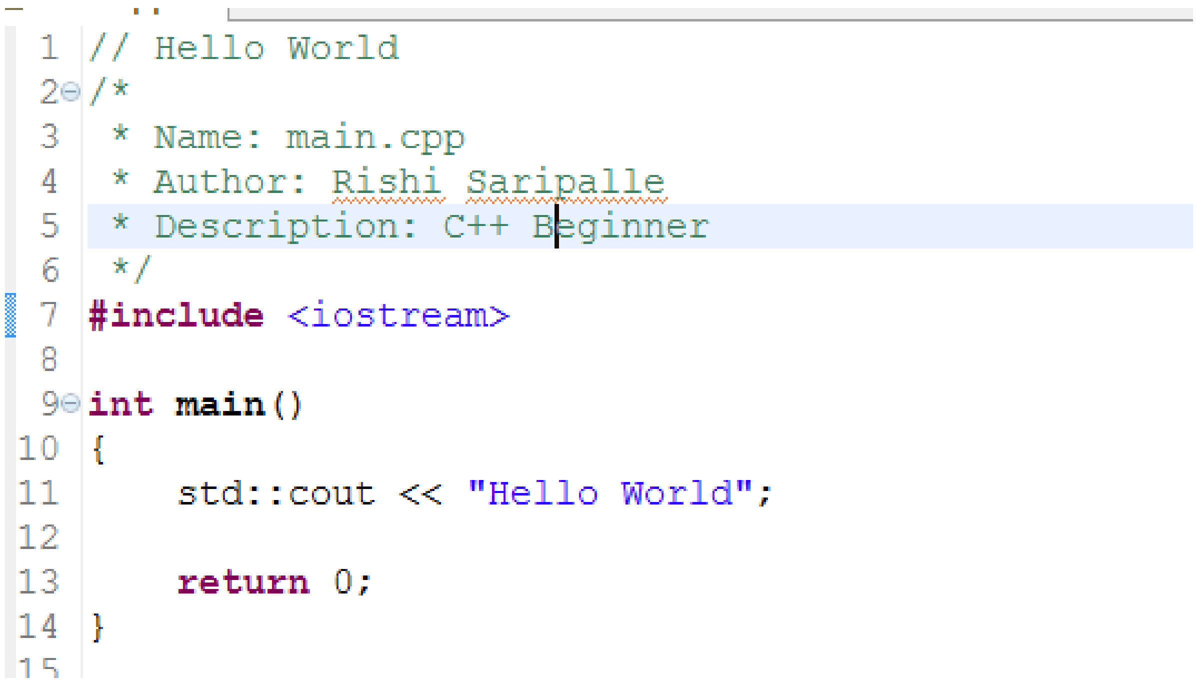
- **C++**
  - Compatible with C
  - Write once, compile anywhere
  - Runs as native executable machine code
  - Pointers, References, and pass-by-value
  - Single and Multiple inheritance
  - Support Templates

- **Java**
  - Provides the Java Native Interface
  - Write once, run anywhere
  - Runs in a virtual machine – JVM
  - Always passed-by-value
  - Singe inheritance
  - Supports Generics

C++ : Hello World !

C++ Program

- Line 1- 7 : // or /* */ comments
- Line 9: # - Directives to be included by the preprocessor
  - *iostream* is a standard directive
- Line 11: Entry point to the program
- Line 13: “::” Resolution Operator
  - cout – (Character Output): Write to default output – console.
  - std – standard
  - **<<** - writes its second argument onto its first.



```
 1 // Hello World
 2 /*
 3  * Name: main.cpp
 4  * Author: Rishi Saripalle
 5  * Description: C++ Beginner
 6  */
 7 #include <iostream>
 8
 9 int main()
10 {
11     std::cout << "Hello World";
12
13     return 0;
14 }
15
```

C++: Libraries & Namespace

Libraries

- C++ has a number of standard libraries. To include a library use ***#include directive***:

  #include <*Library_Name*>

  #include <*iostream*> #include <*cmath*> #include <*ctime*>

- For more information on different libraries
  - https://en.wikipedia.org/wiki/C%2B%2B_Standard_Library

C++: Compilation Process

What happens

- Preprocessing
  - Preprocessor handles the directives, like #include or #define.
  - Replaces #include directives with the content of the respective files
- Compilation
  - Performed on each output of the preprocessor
  - Parses the C++ code and produces an **Object** file
    - Three separate C++ files, you will have three object files <filename>.o or <filename>.obj
- Linking
  - **Object** file generated are linked together with the **Object** files for any library functions to produce an **executable**

http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

C++: Identifiers

Identifier or Variables

- Starts with a letter
- “_” can be first
  - Reserved for compiler-specific keywords or external identifiers
- Sequence of one or more letters, digits, or underscore characters (_)
- Spaces, punctuation, and symbols cannot be part of an identifier
- Avoid Keywords
  - Specific compilers may also have additional specific reserved keywords.
- C++ language is a "case sensitive" language.
  - **RESULT** different from **result** from **Result** from **reSult**

C++: Datatypes

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as char. At least 8 bits.
	signed **short** *int*	Not smaller than char. At least 16 bits.
	signed **int**	Not smaller than short. At least 16 bits.
	signed **long** *int*	Not smaller than int. At least 32 bits.
	signed **long long** *int*	Not smaller than long. At least 64 bits.

Group	Type names*	Notes on size / precision
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

C++: Declaring Variables

- **How is it done ?**

  int count; double value;

  - In general,
    - ***Type** variable_name;*
  - Good Practice:
    - Always initialize the variables

C++: Operators

- **Operators**
  - Arithmetic Operations
    - +, -, *, /, %, =,
    - += , x+=y same as x = x+y
    - -=, *=, /=, %=
  - Increment & Decrement Operators
    - ++, --
    - Prefix = ++x
      - If x = 5 and y = ++x;
    - Postfix = x++
      - If x =5 and y = x++;

- Relational Operators
  - >,>=,<,<=, ==, !=
- Logical Operators
  - !, &&, ||

C++: Operators

Operators

» Bitwise Operators

Operator	Symbol	Form	Explanation	Example
left shift	<<	x << y	all bits in x shifted left y bits	2<<1 = 4
right shift	>>	x >> y	all bits in x shifted right y bits	2>>1 = 1
bitwise NOT	~	~x	all bits in x flipped	~2 = -3
bitwise AND	&	x & y	each bit in x AND each bit in y	2 & 1 = 0
bitwise OR	|	x | y	each bit in x OR each bit in y	2 | 1 = 3
bitwise XOR	^	x ^ y	each bit in x XOR each bit in y	2^ 1 = 3

C++: Basic Input & Output

Refer to: iostream

- **iostream**
  - “>>” – the input operator
  - cin – standard input stream
    - Ignores any character(s) after a whitespace
  - “<<“ – the output operator
  - cout – standard output stream
  - cerr – standard error reporting stream

```
#include <iostream>
using namespace std;
int main(){
    int value;
    cout<<"Enter a value: "<<endl;
    cin>>value;
    cout<<"Entered value is: "<<value<<endl;
}
```

C++: Scope

Refer to: identifiers

Variable Scope

- Local
  - Only accessible in the defined block
  - Expires when the block is executed
- Global
  - Internal
    - Only accessible in a given file
  - External
    - Accessible anywhere in the project workspace
    - Achieved using extern keyword

C++: Scope

Refer to: identifiers

Scope

- Global Variables
  - They are not thread safe
    - i.e. they are not synchronized
- Constant Variables
  - Variables whose value cannot be changed
  - Achieved using const keyword

```
double const PI = 3.141;
```

C++: sizeof

sizeof

- Obtains the number of bytes occupied by a *type*, or by a *variable*
- Example

```
int height = 74;
cout << "The height variable occupies " << sizeof(height) << " bytes." <<endl;
cout << "Type \"long long\" occupies " << sizeof (long long) << " bytes." <<endl;
cout << "The expression height*height/2 occupies - "<< sizeof (height*height/2) << " bytes." <<endl;
```

C++: String Library

Refer to: String

- **String**
  - Library
    - **#include <string>**
  - Declaration
    - string name;
    - string name (“Rishi")

C++: String & Chars

Refer to: CString

Strings

» C-string – character array terminated by **‘\0’**

```
char nameChar[] ="C++";
```

C	+	+	\0

```
nameChar[0] = ‘c’

nameChar[2] = ‘+’
```

```
char nameChar[] ="C++";
```

C++: Function

Function

- Captures and Performs a small unit of work
- How to write the function

```
return_type functionName (parameters){
        .....
        .......
}
```

- Example

```
int multiply (int x, int y){
        return x*y;
}
```

- How to use the function

```
multiply(10,20)
```

C++: Function

Function

- Using the function
  - Two Options
    - Option 1 – define the function before using it
    - Option 2 - Forward Declaration
      - **Declare** the function before you can use it, but not its definition
      - Notifies the complier that the function exists with
    - How to write a declaration
      - Just the first line of the function without its body.

Refer to: Function

C++: Function

- **Pass by value**
  - The argument’s value is copied into the function’s parameter
    - Pros
      - Arguments are never changed by the function being called
      - Can be variables (e.g. x), literals (e.g. 6), expressions (e.g. x+1), structs & classes, and enumerators
    - Cons
      - Copying large data entities (structs and classes) can incur a significant performance penalty
        - Also, bad programming practice.

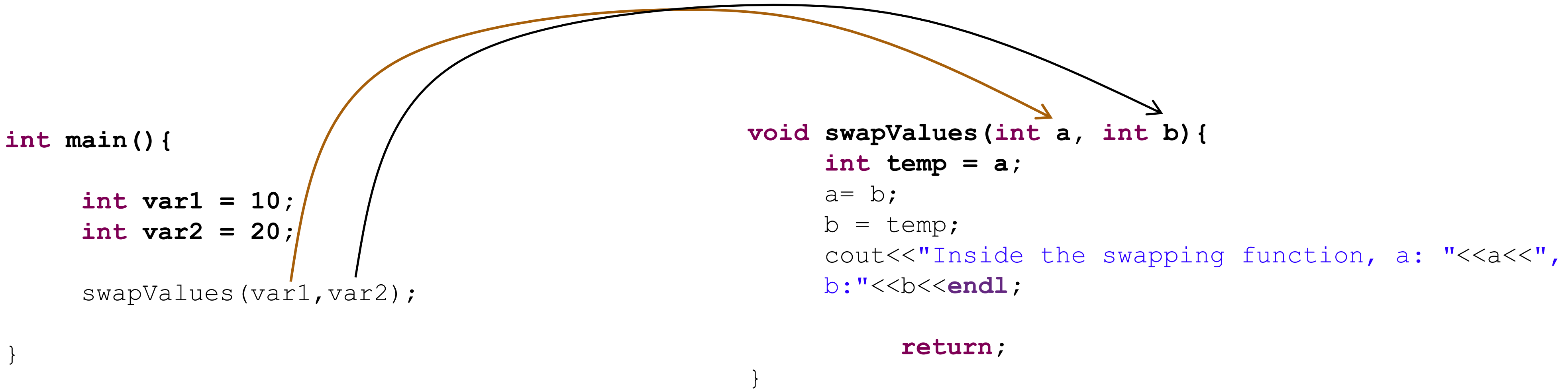
C++: Function

Refer to: FunctionExample2

— Pass by value

»

The value of var1 (10) is assigned/copied to *a*
and the value of var2 (20) is assigned/copied to b



```
int main(){

    int var1 = 10;
    int var2 = 20;

    swapValues(var1,var2);

}
```

```
void swapValues(int a, int b){
    int temp = a;
    a= b;
    b = temp;
    cout<<"Inside the swapping function, a: "<<a<<",
    b:"<<b<<endl;

        return;
}
```

C++: Function

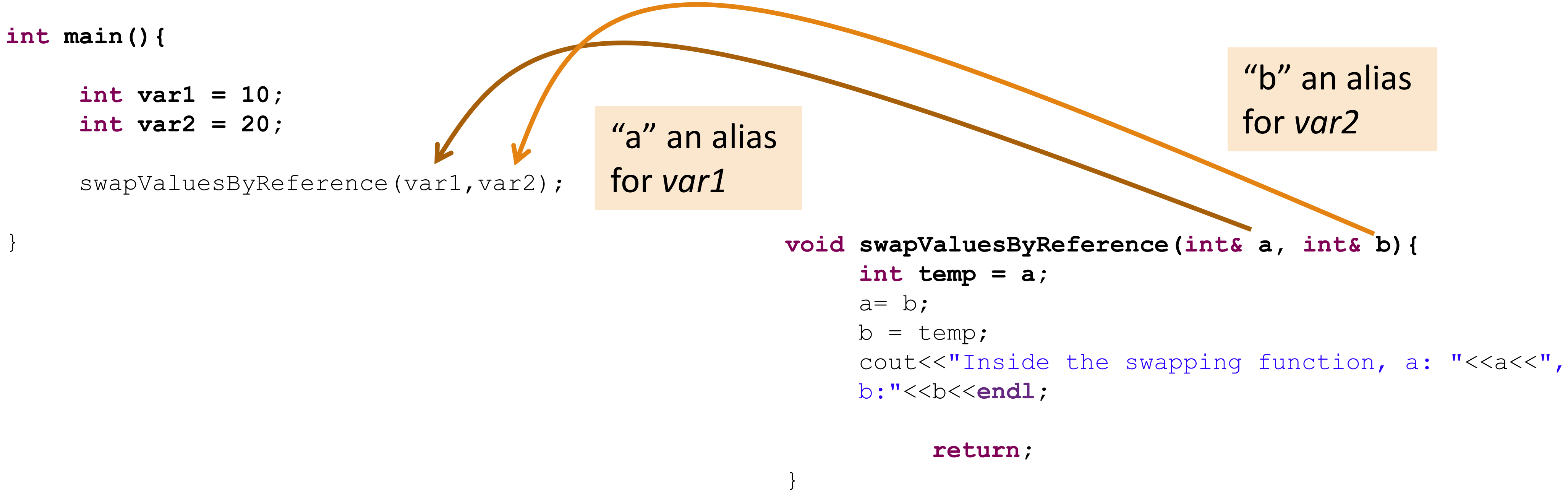
Pass by Reference

- Pass by reference
  - The argument’s **reference** is passed into the function’s parameter
    - & argument -> reference to argument variable
  - Pros
    - Performance improvement and efficient

C++: Function

Refer to: Functions2

— Pass by Reference



```
int main(){

     int var1 = 10;
     int var2 = 20;

     swapValuesByReference(var1,var2);

}
```

“a” an alias for *var1*

“b” an alias for *var2*

```
void swapValuesByReference(int& a, int& b){
     int temp = a;
     a= b;
     b = temp;
     cout<<"Inside the swapping function, a: "<<a<<",
     b:"<<b<<endl;

          return;
}
```

C++: References

— Reference

```
int main(){

    int var1 = 10;
    int var2 = 20;

    /*
     * Reference always needs to initialized to a variable. Otherwise, you will see error –
                   error: ‘var2Ref’ declared as reference but not initialized  int& var2Ref;
     */
    int& var2Ref; // Not valid
    int& var2Ref = var2; // valid

}
```

C++: Function

Overloading

- Having multiple function definition with the same ***name***
- **Provided**, you have different *number* or *types* of parameters

```
void swapValues(int a, int b){
     …….
     ….. 
}
// Different set of types
void swapValues(double a, int b){
     …….
     ….. 
}
// Different number of parameters
void swapValues(int a, int b, int c){
     …….
     ….. 
}
```

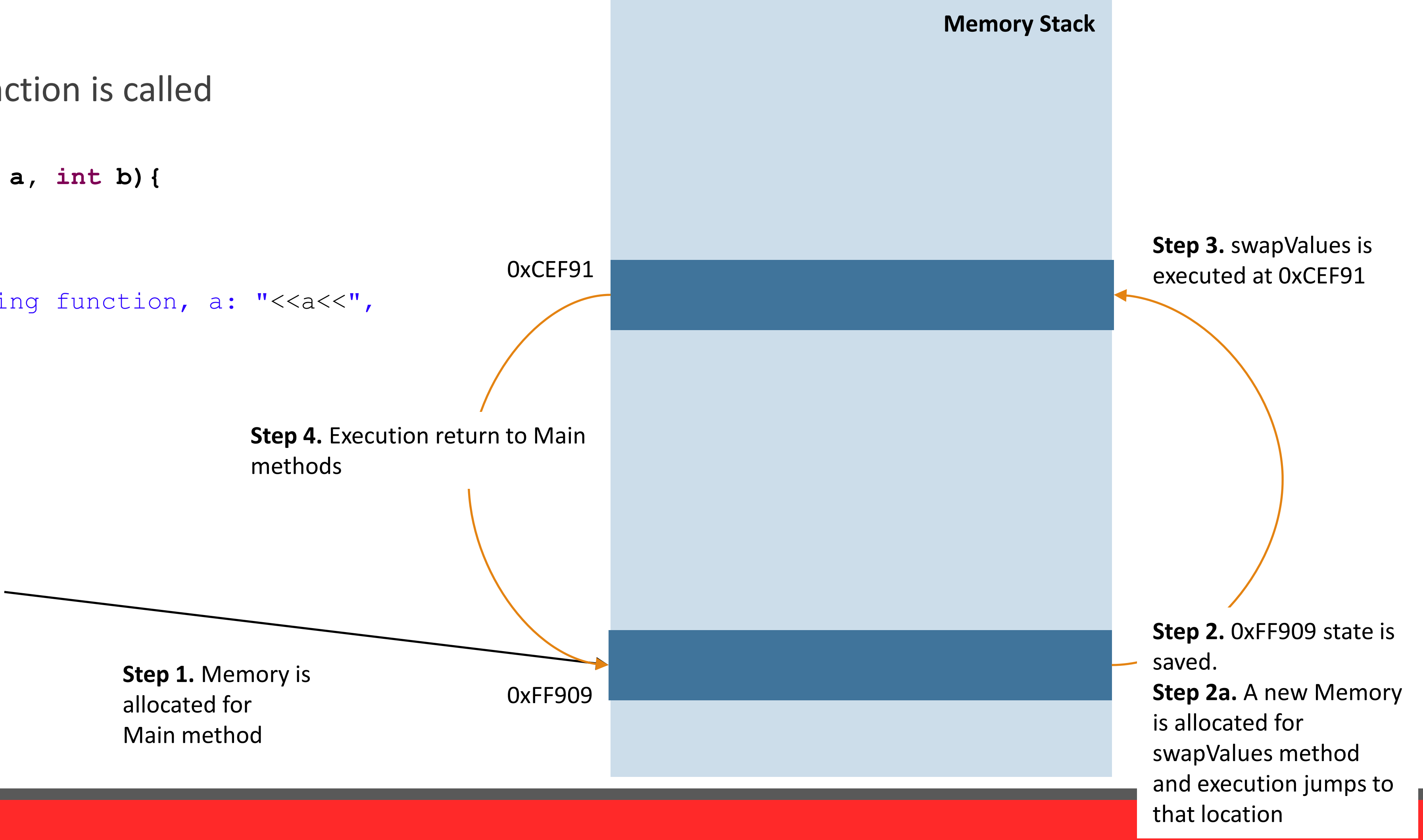
```
// NOT a valid overloading. Return type doesn’t matter when
overloading functions
double swapValues(int a, int b){
     ….. 
}
double swapValues(const int a, int b){
     ….. 
}
void swapValues(const int& a, int b){
     …….
     ….. 
}
```

C++: Function

Inline Function

- What happens when a function is called

```
void inline swapValues(int a, int b){
     int temp = a;
     a= b;
     b = temp;
     cout<<"Inside the swapping function, a: "<<a<<",
     b:"<<b<<endl;

          return;
}

int main(){

     int var1 = 10;
     int var2 = 20;

     swapValues(var1,var2);

}
```



Step 1. Memory is allocated for Main method

Step 2. 0xFF909 state is saved.
Step 2a. A new Memory is allocated for swapValues method and execution jumps to that location

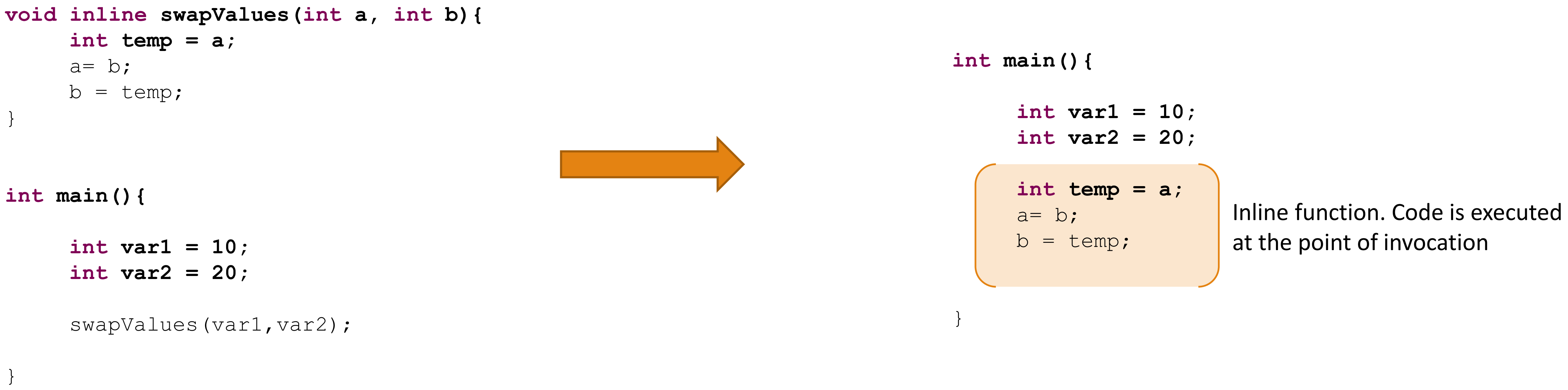
Step 3. swapValues is executed at 0xCEF91

Step 4. Execution return to Main methods

C++: Function

Inline Function

- Compiler executes the function code at the point of invocation
  - Instead of going to the function for execution, the code defined in the function is executed at the point of invocation.



```
void inline swapValues(int a, int b){
     int temp = a;
     a= b;
     b = temp;
}


int main(){

     int var1 = 10;
     int var2 = 20;

     swapValues(var1,var2);

}
```

```
int main(){

     int var1 = 10;
     int var2 = 20;

     int temp = a;
     a= b;
     b = temp;

}
```

Inline function. Code is executed at the point of invocation

C++: Arrays

C++: Array

- **Arrays**
  - Initialization

    ```
    int number[10];
    int number[] = {0,1,2,3,4,5};
    ```

  - Access

    ```
    int var = number[0]; index from 0 and last index is size -1
    number[2] = 222; assign value using index
    ```

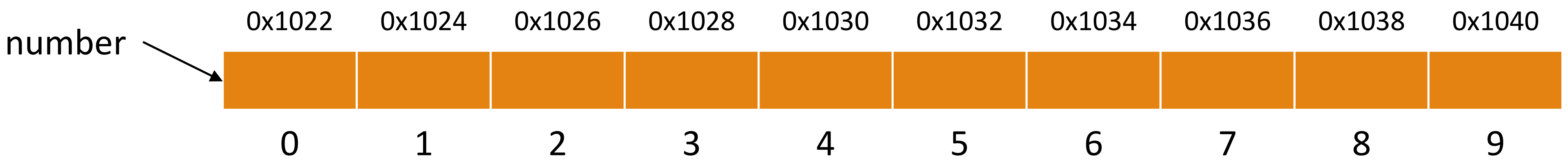
Take Home:

Why does index start from ‘0’

C++: Array

Arrays

- What happens when you initialize an array ? int number[10];
  - Three things will be remembered by the compiler
    - Address location of ‘0th’ index
    - Type of the Array
    - Size of the Array



number

0x1022	0x1024	0x1026	0x1028	0x1030	0x1032	0x1034	0x1036	0x1038	0x1040
0	1	2	3	4	5	6	7	8	9

C++: Array

Refer to: Array

— Arrays

» Store values into an Array

```
for(int i=0;i<10;i++){
        cout<<"Enter value for numbers["<<i<<"]: ";
        cin>>numbers[i];
        cout<<endl;
}
```

» Access Values

```
for(int i=0;i<10;i++){
        cout<<numbers[i]<<" ";
}
```

C++: Array

Refer to: Array

Array as function argument

Pass the memory address of the 0th index → random[0]

Always pass in the size as another argument

```
int main(){
    srand(time(NULL));
    int random[10];

    fillArray(random,10);

    cout<<"Random values"<<endl;
    for(int i:random)
        cout<<i<<" “;
    }
```

```
void fillArray(int a[],int size){
          for(int i=0;i<size;i++)
                a[i]=rand()%20;

}
```

time() is in ctime library

C++: Multidimensional Array

Multidimensional Arrays

- Initialization

```
int numbers[10][20];  // 10 rows and 20 column
int numbers3D[10][20][10]; // size is 10 x 20 x 10
int numbers[3][5] = {
                { 1, 2, 3, 4, 5, },
                { 6, 7, 8, 9, 10, },
                { 11, 12, 13, 14, 15 } };
 int numbers[][] =  {
                { 1, 2, 3, 4 },
                { 5, 6, 7, 8 } };
 int array[3][5] = { 0 }
```

- Access

```
cout<< numbers[0][0] // numbers[row][col];
cout<< numbers3D[0][1][0] // numbers[row][col][index];
```

C++: const modifier

Refer to: Array

What does it do?

- Sometime, you want to make sure the arguments are READ only
  - i.e. the function can only use the value, but not change it
- const will make sure the arguments are read only
  - Any attempt of changing it will throw errors

```
double getAverage(const int a[], const int size){
        int sum = 0;
        for(int i=0;i<size;i++){
                sum+=a[i];
        }
        return sum/size;
}
```

Thank You

Question, Comments & Feedback