C++ Programming: Session 1

Anupama Chandrasekhar (Aug 2 2018)

Credits: http://realmeneatplants.com/post/148306356584/a-journey-of-a-thousand-miles-begins-with-a-single

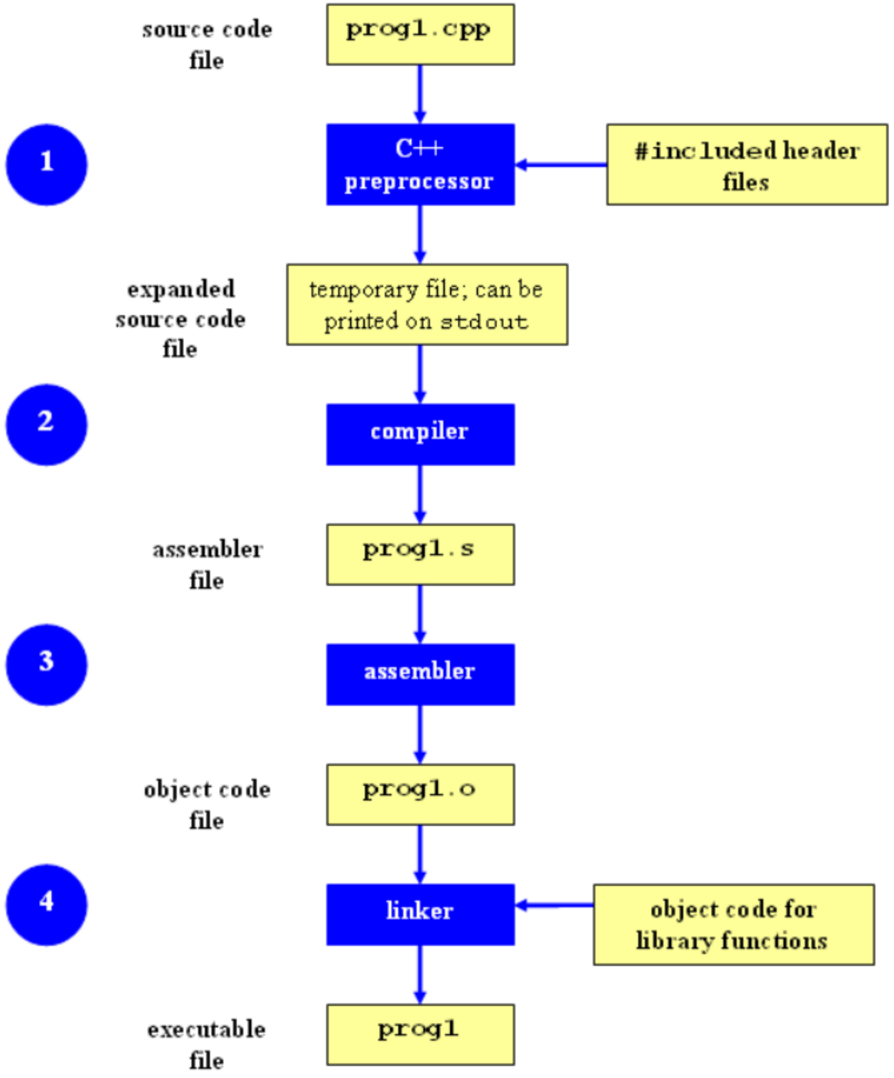
Course Agenda

- Introduction
- History of C++
- The Compilation Process
- Program Structure
- Built-in types, Operators and type conversions
- Control Flow
- Pointers, Arrays and References

Some History of C++

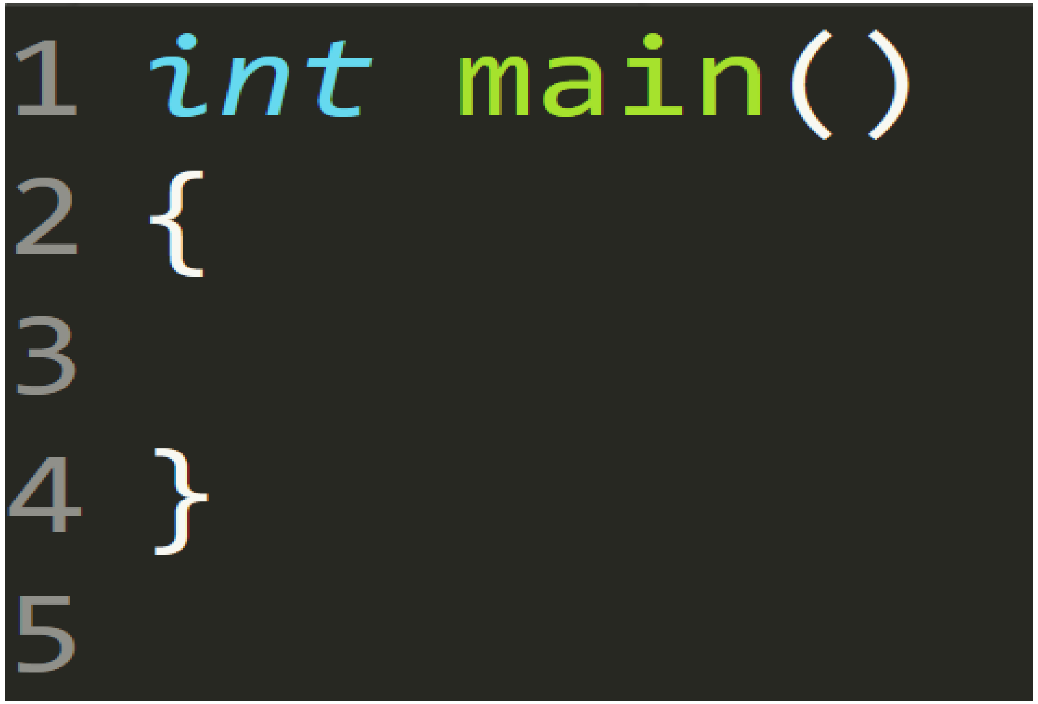
- C++ was created by Bjarne Stroustrup by adding some object oriented programming paradigms from Simula to C. These additions included classes, basic inheritance, inlining, default function arguments and strong type checking.
- More and more OOP like virtual function, function overloading, references, multiple inheritance etc.. were added over time
- After about 10 years of silence, C++ got a major reboot in 2011, auto keyword, range-for, new container classes, array initialization lists, variadic templates, atomics, threading support, regular expression support.
- C++ continues to evolve, C++14, C++17 and work is underway for C++20.

C++ Basics: The Compilation Process



- Preprocessor: Takes in the C++ source code and processes the line beginning with “#”, preprocessor directives. Macro substitution happens after this.
- Compilation: Cpp Code->Machine Code (object file)
- Linker: Link multiple object files into an executable

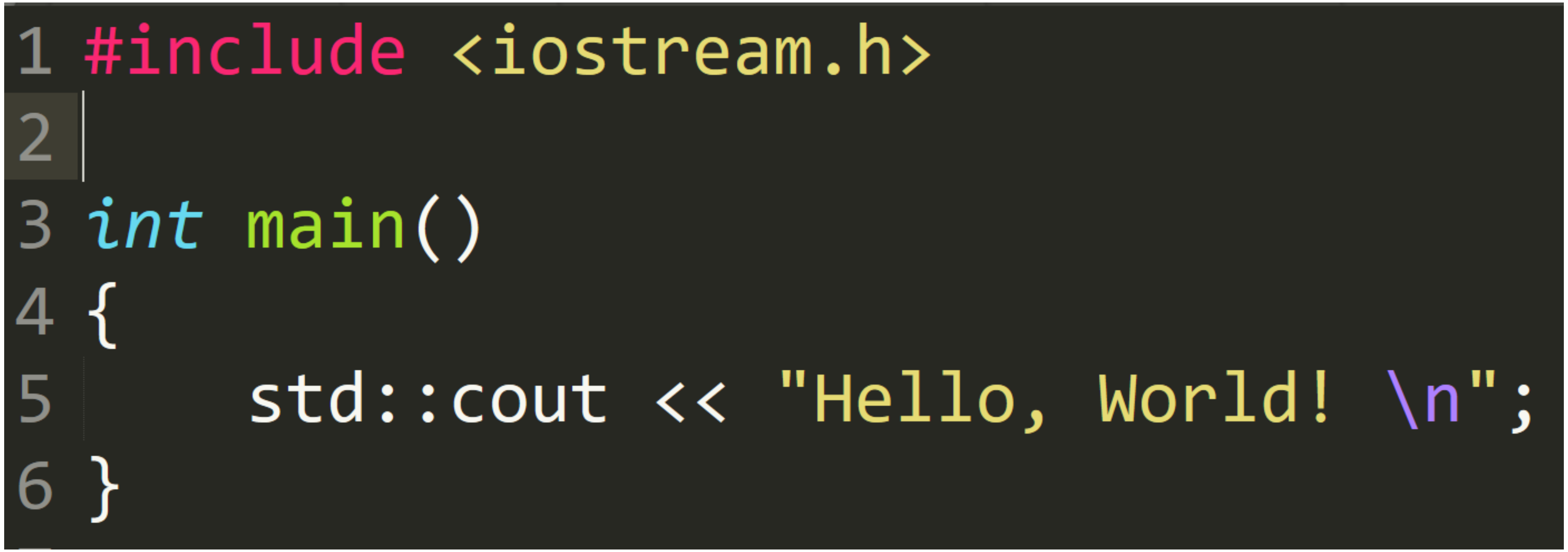
C++ Basics: The Minimal Program



```
int main()
{

}

```

- Every C++ program has exactly one global main function, the program starts by executing that function, int retuned is the program’s return value to the system

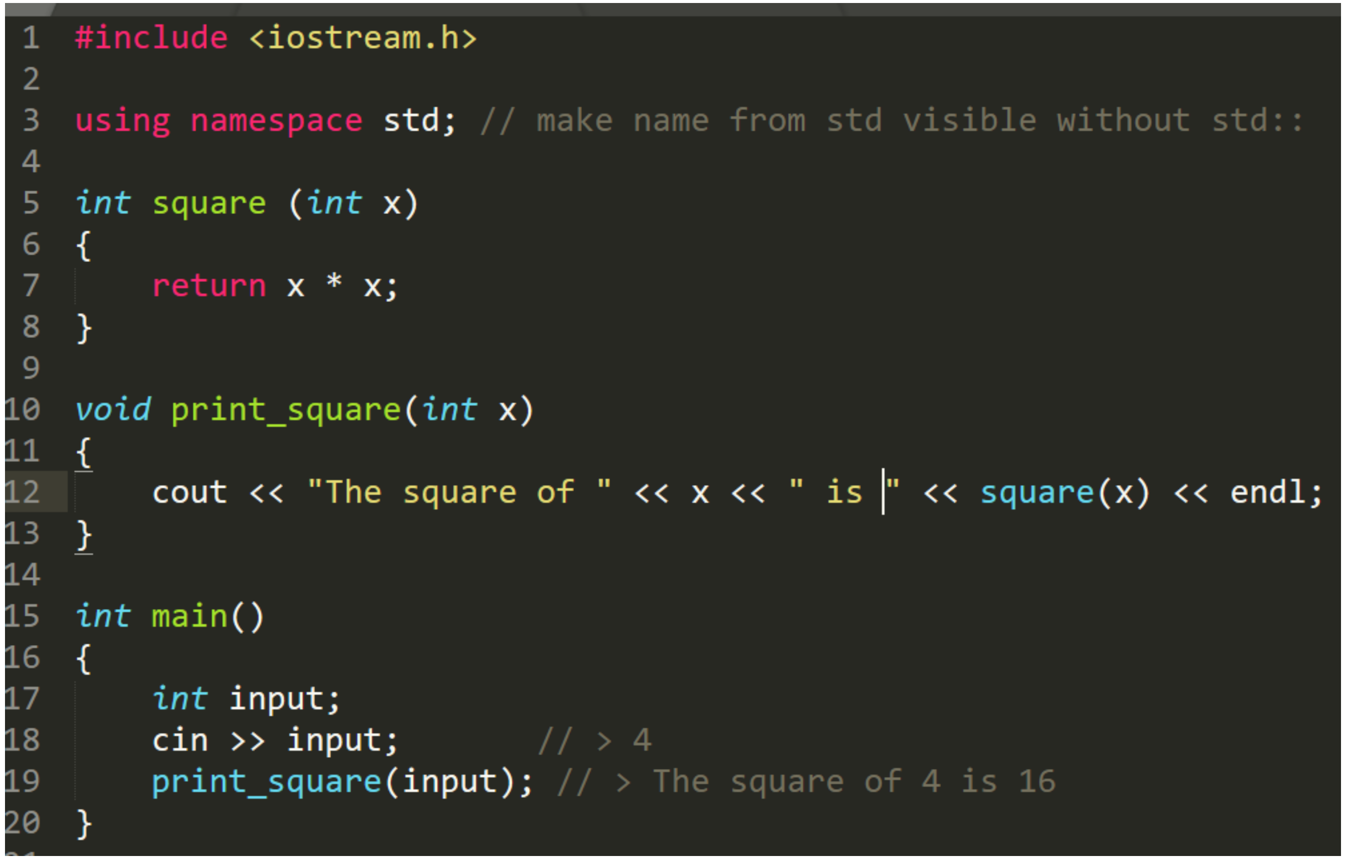
C++ Basics: Hello, World!



```cpp
#include <iostream.h>

int main()
{
    std::cout << "Hello, World! \n";
}
```

- “Hello, World!” is a string literal that is written to standard output stream cout, which is in the standard library namespace. “\n” is the newline character

C++ Basics: Functions

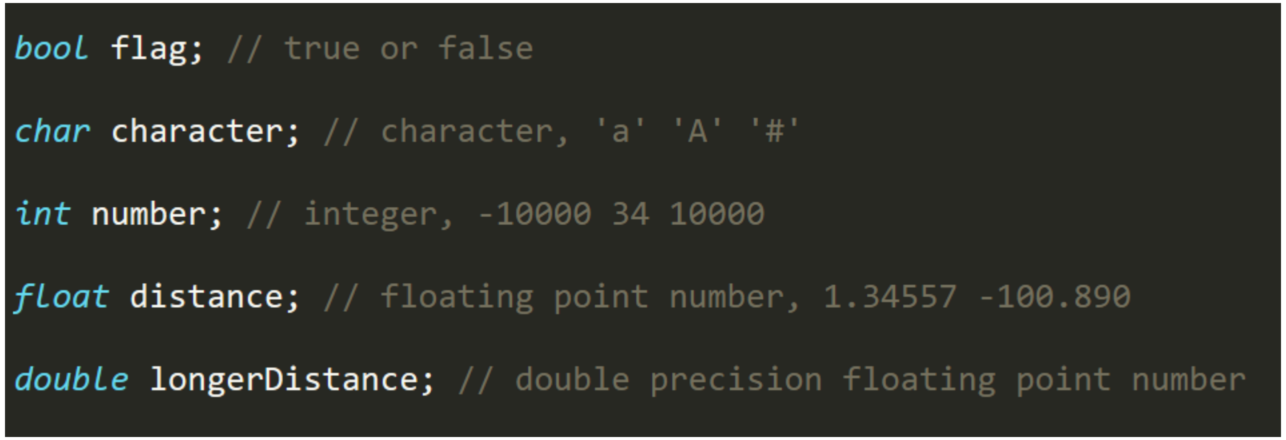


```cpp
#include <iostream.h>

using namespace std; // make name from std visible without std::

int square (int x)
{
    return x * x;
}

void print_square(int x)
{
    cout << "The square of " << x << " is " << square(x) << endl;
}

int main()
{
    int input;
    cin >> input;       // > 4
    print_square(input); // > The square of 4 is 16
}
```

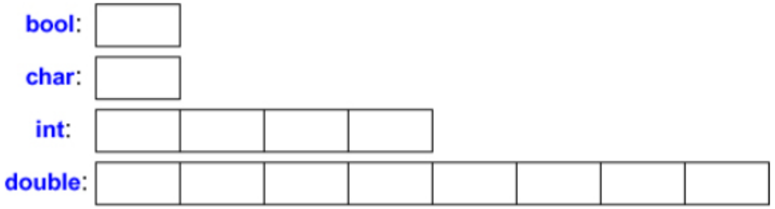
- All executable code is called, directly or indirectly from main.

C++ Basic: Built in Types

- Every variable/expression has a type that determines the operations that may be performed on it.



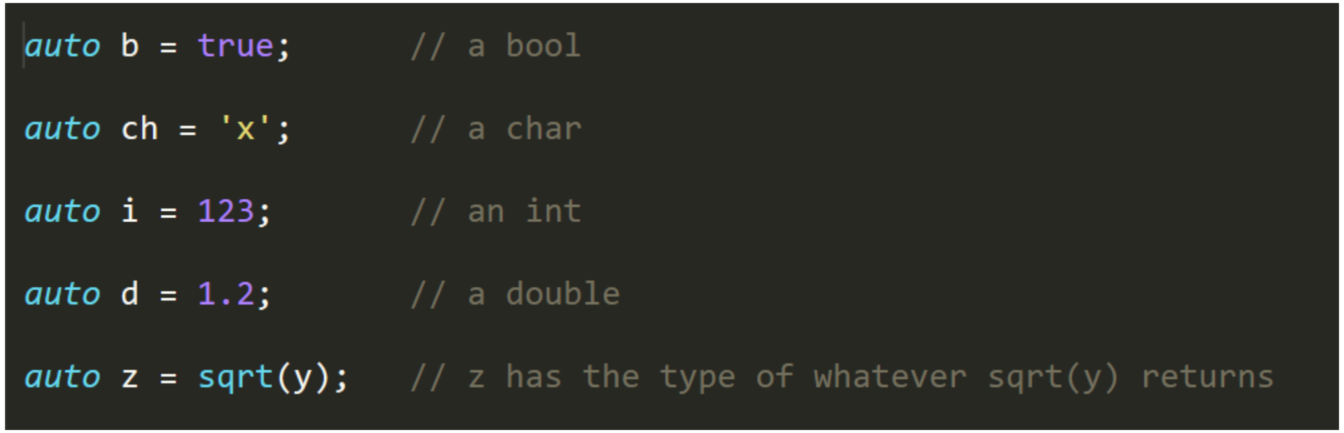
```
bool flag; // true or false

char character; // character, 'a' 'A' '#'

int number; // integer, -10000 34 10000

float distance; // floating point number, 1.34557 -100.890

double longerDistance; // double precision floating point number
```



- Each fundamental type has a fixed size for a certain implementation and that determines the range of values that can be stored in it. The size of a type can be obtained by the `sizeof()` operator, e.g: `sizeof(char)` is typically 1.

C++ Basics: Auto

- When defining a type you don’t have to explicitly state the type it can be deduced by the compiler from the initializer – say hello to “auto” introduced in C++11

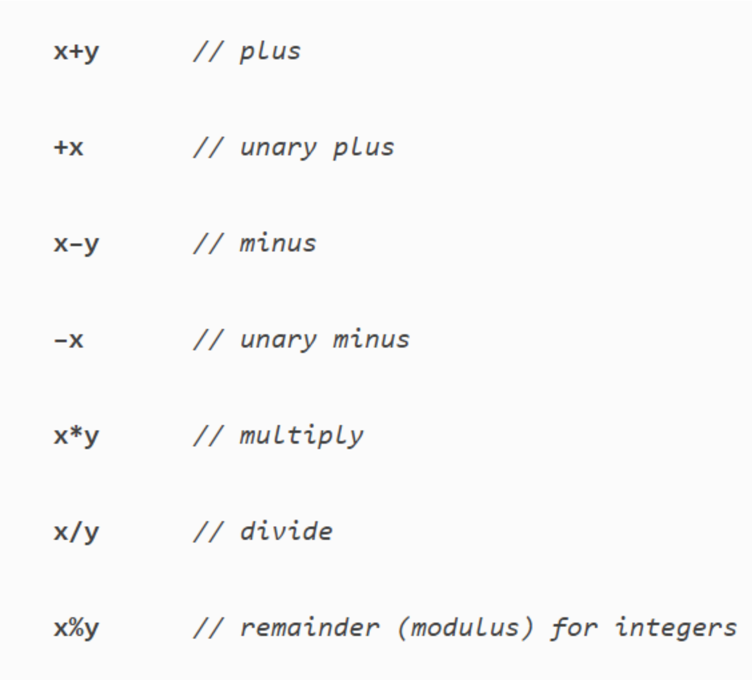


```
auto b = true;      // a bool

auto ch = 'x';      // a char

auto i = 123;       // an int

auto d = 1.2;       // a double

auto z = sqrt(y);   // z has the type of whatever sqrt(y) returns
```

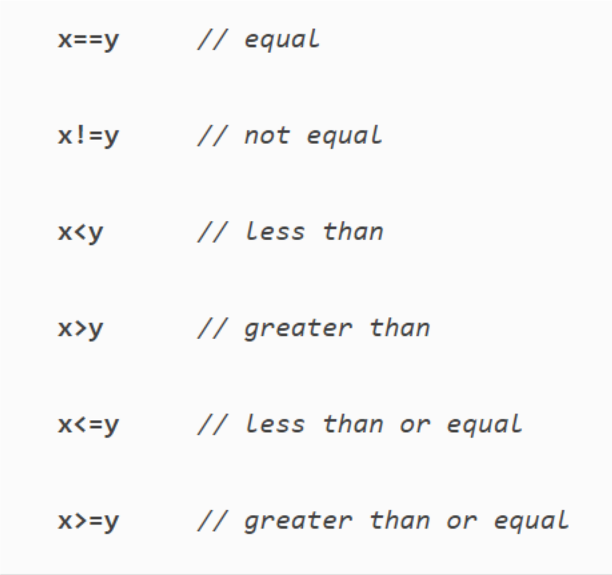
- Note: With auto we use = because there is no type conversion. You may choose not to use auto to be more explicit about intended precision etc.

C++ Basics: Operators

- Common operations



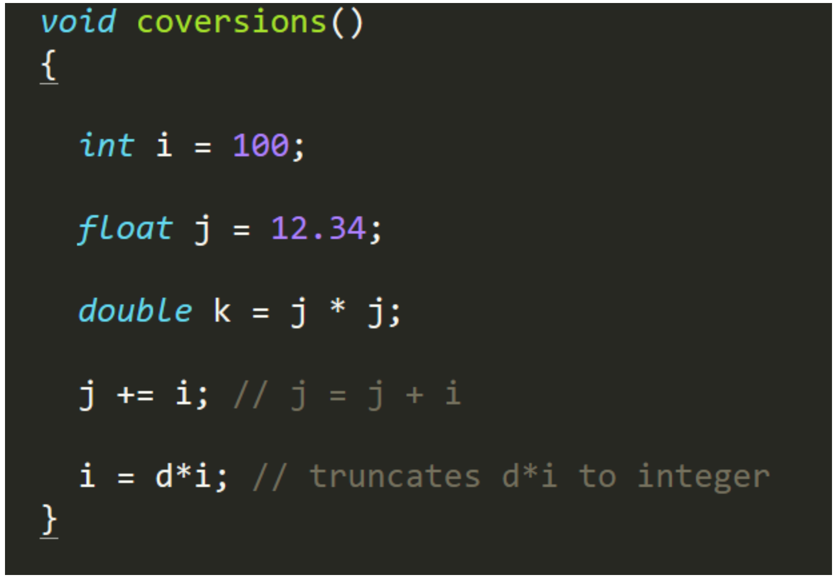
```
x+y       // plus

+x        // unary plus

x-y       // minus

-x        // unary minus

x*y       // multiply

x/y       // divide

x%y       // remainder (modulus) for integers
```



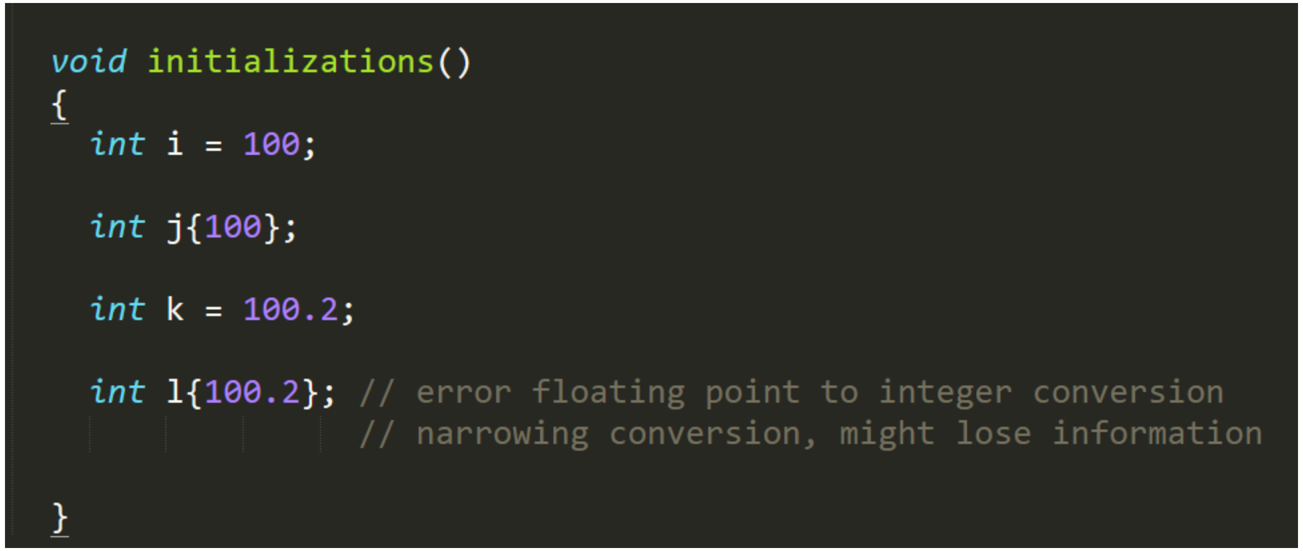
```
x==y      // equal

x!=y      // not equal

x<y       // less than

x>y       // greater than

x<=y      // less than or equal

x>=y      // greater than or equal
```

C++ Basics : Conversions

- Implicit conversions (when meaningful) and initializations



```cpp
void coversions()
{
  int i = 100;

  float j = 12.34;

  double k = j * j;

  j += i; // j = j + i

  i = d*i; // truncates d*i to integer
}
```



```cpp
void initializations()
{
  int i = 100;

  int j{100};

  int k = 100.2;

  int l{100.2}; // error floating point to integer conversion
                // narrowing conversion, might lose information

}
```

- Introducing auto: use "`auto`" when the type can be deduced unambiguously. Use to avoid redundancy and long type names esp in generic programming

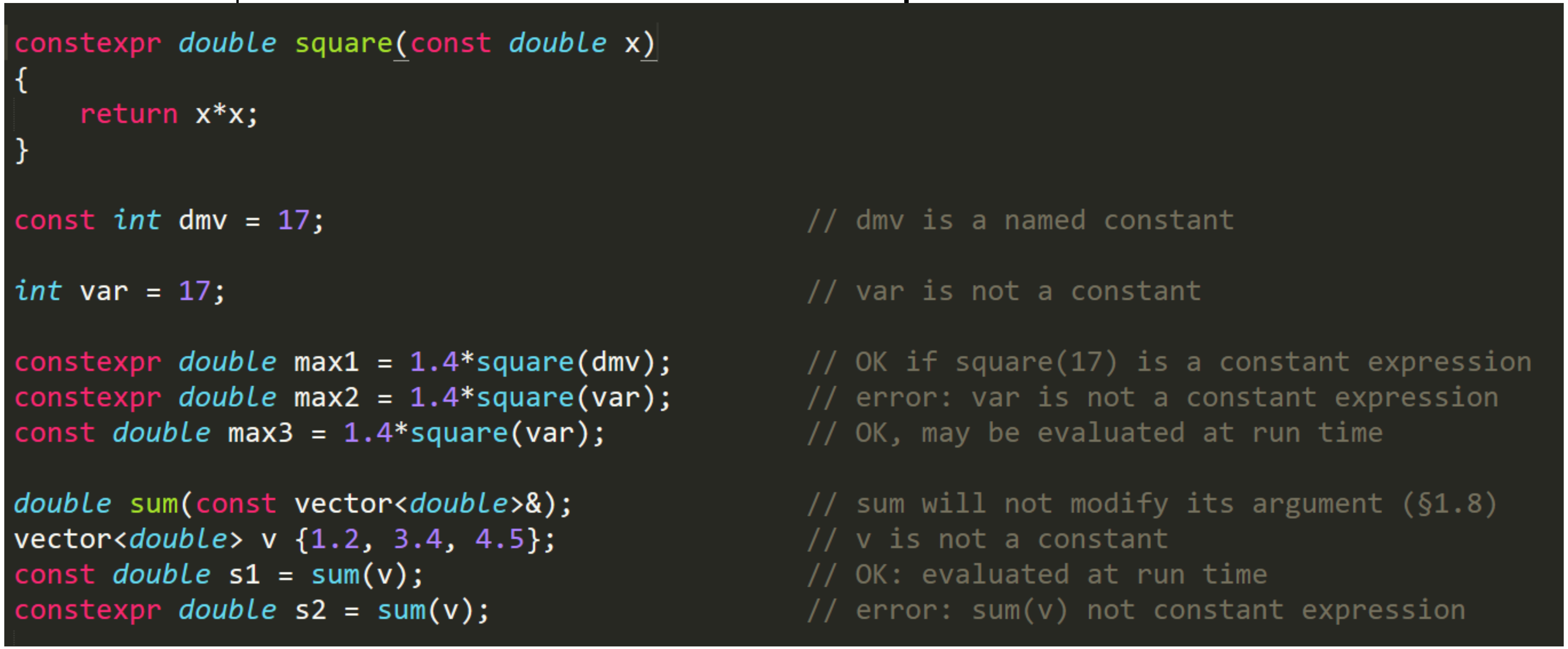
C++ Basics: Operators and type conversions (Exercise)

- **Exercise Problem : Computing Volume (2_TypeConversions.cpp)**

- **Expected Result :**
  - **157.464 52.9854 857.985**

C++ Basics : Constants

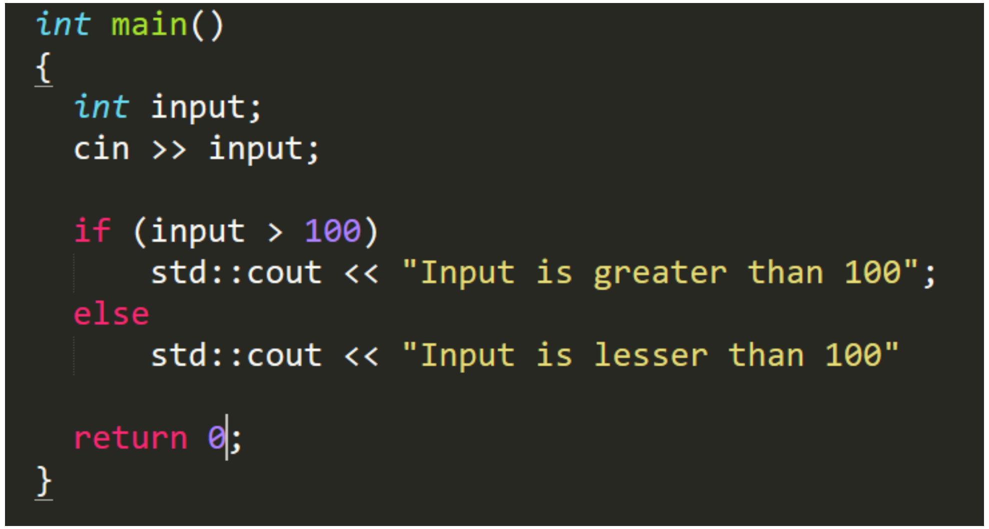
- Two notions of immutability:
  - “const” : I promise not to change this value
  - “constexpr”: ~ To be evaluated at compile time.



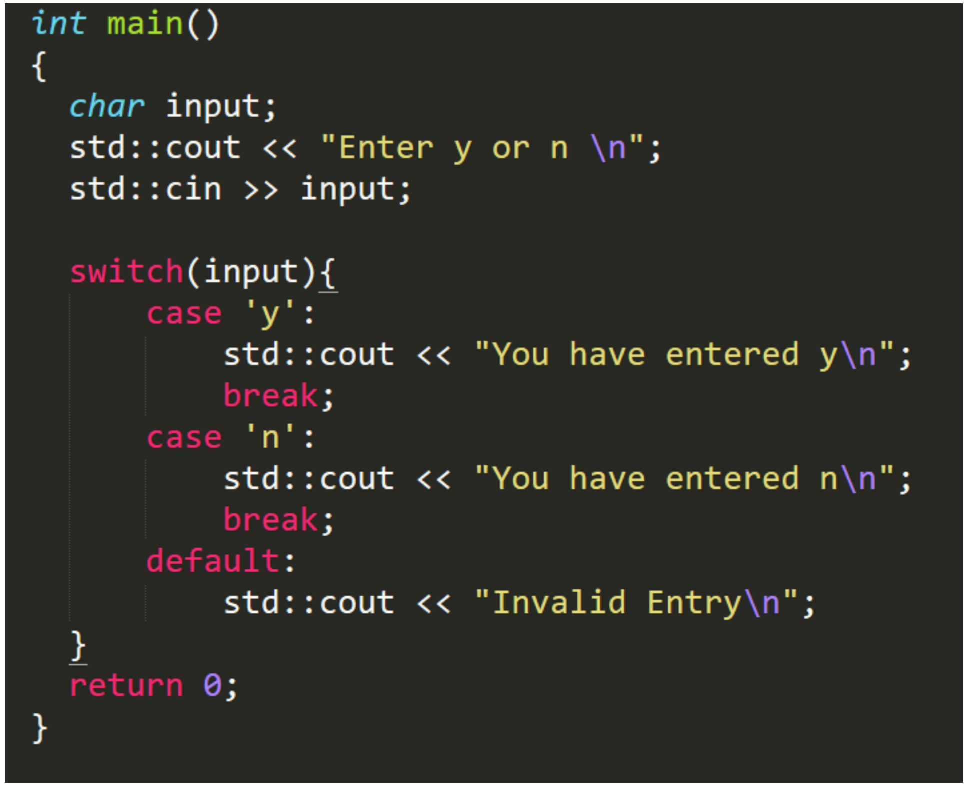
```
constexpr double square(const double x)
{
    return x*x;
}

const int dmv = 17;                             // dmv is a named constant

int var = 17;                                   // var is not a constant

constexpr double max1 = 1.4*square(dmv);        // OK if square(17) is a constant expression
constexpr double max2 = 1.4*square(var);        // error: var is not a constant expression
const double max3 = 1.4*square(var);            // OK, may be evaluated at run time

double sum(const vector<double>&);              // sum will not modify its argument (§1.8)
vector<double> v {1.2, 3.4, 4.5};               // v is not a constant
const double s1 = sum(v);                       // OK: evaluated at run time
constexpr double s2 = sum(v);                   // error: sum(v) not constant expression
```

C++ Basics: Control Flow

- As opposed to straight line code, control flow adds the ability to take different code paths based on a condition



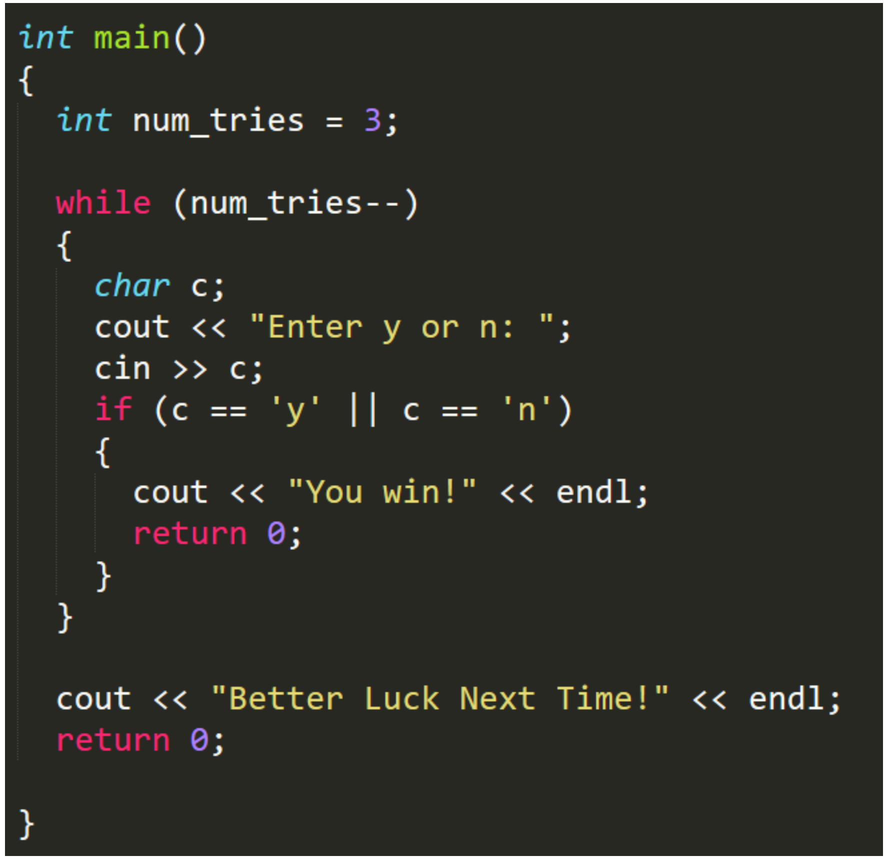
```
int main()
{
  int input;
  cin >> input;

  if (input > 100)
      std::cout << "Input is greater than 100";
  else
      std::cout << "Input is lesser than 100"

  return 0;
}
```



```
int main()
{
  char input;
  std::cout << "Enter y or n \n";
  std::cin >> input;

  switch(input){
      case 'y':
          std::cout << "You have entered y\n";
          break;
      case 'n':
          std::cout << "You have entered n\n";
          break;
      default:
          std::cout << "Invalid Entry\n";
  }
  return 0;
}
```

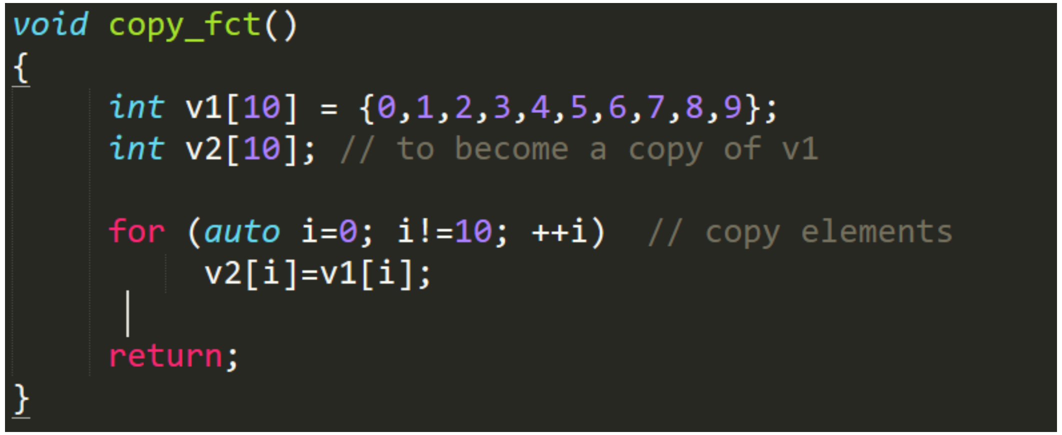
C++ Basics: Loops

- while, do-while, for-loop : Execute until the condition is false

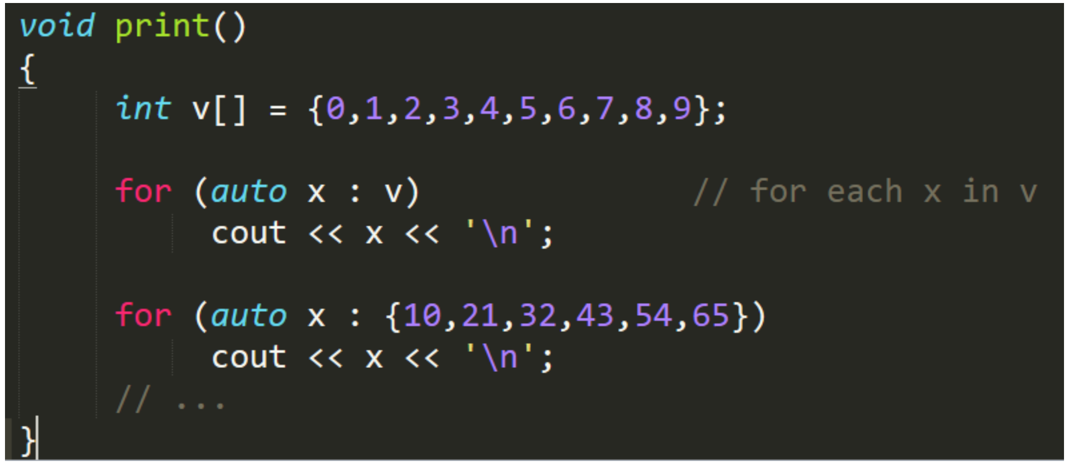


```
int main()
{
  int num_tries = 3;

  while (num_tries--)
  {
    char c;
    cout << "Enter y or n: ";
    cin >> c;
    if (c == 'y' || c == 'n')
    {
      cout << "You win!" << endl;
      return 0;
    }
  }

  cout << "Better Luck Next Time!" << endl;
  return 0;

}
```

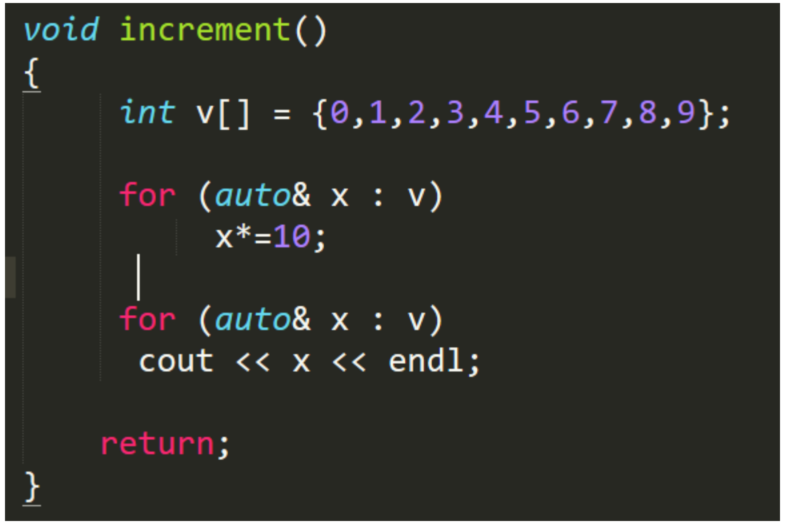
C++ Basics: Loops (`for`)



```cpp
void copy_fct()
{
     int v1[10] = {0,1,2,3,4,5,6,7,8,9};
     int v2[10]; // to become a copy of v1

     for (auto i=0; i!=10; ++i)  // copy elements
          v2[i]=v1[i];

     return;
}
```



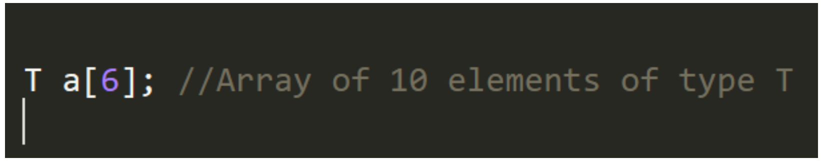
```cpp
void print()
{
     int v[] = {0,1,2,3,4,5,6,7,8,9};

     for (auto x : v)               // for each x in v
          cout << x << '\n';

     for (auto x : {10,21,32,43,54,65})
          cout << x << '\n';
     // ...
}
```



```cpp
void increment()
{
     int v[] = {0,1,2,3,4,5,6,7,8,9};

     for (auto& x : v)
          x*=10;

     for (auto& x : v)
      cout << x << endl;

    return;
}
```

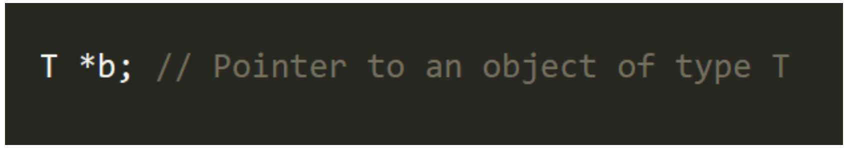
C++ Basics: Pointers and Arrays

- Arrays of type T are declared as:



```
T a[6]; //Array of 10 elements of type T
```

- Pointers of type T are declared as:

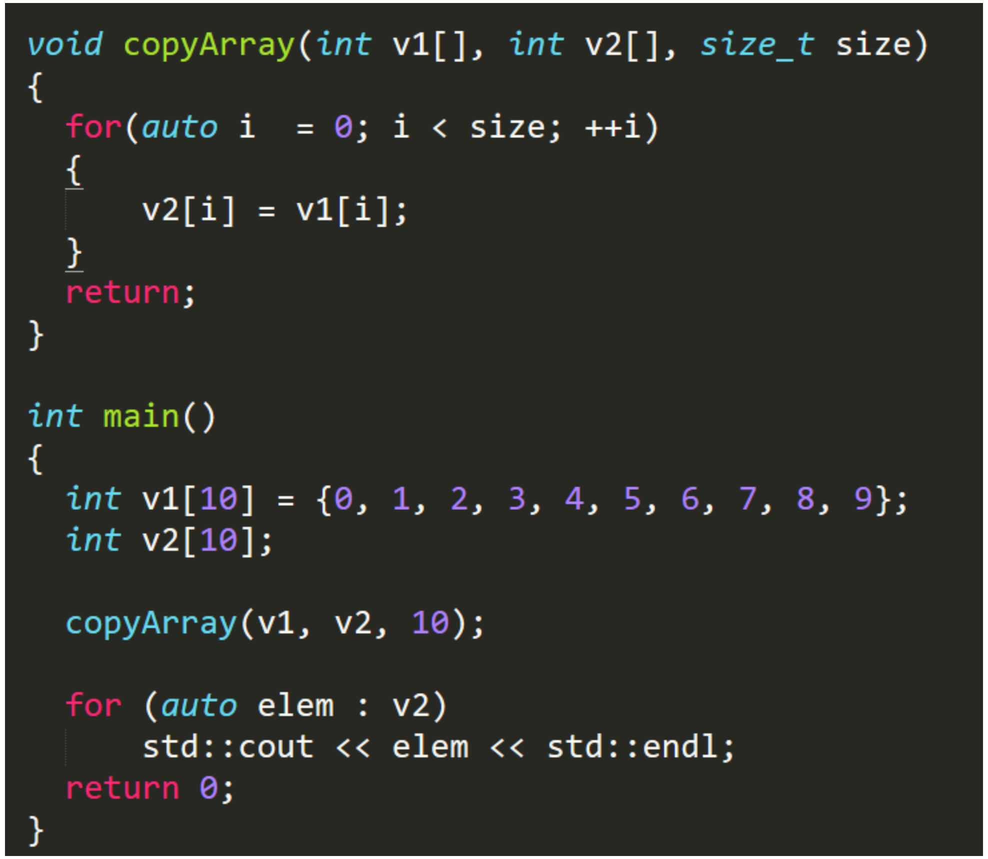


```
T *b; // Pointer to an object of type T
```

- Terminology: [] means “array of”, * means “pointer to. All arrays have 0 as their lower bound, the size of an array must be a constant expression. A pointer variable hold the address of the object it is pointing to

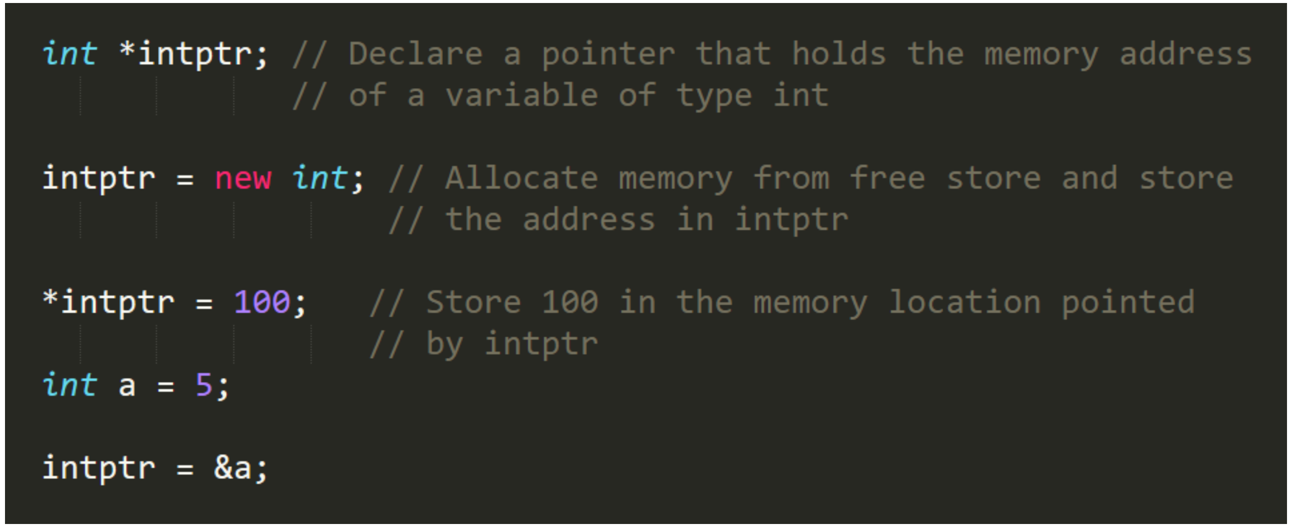
C++ Basics : Array (Examples)

- Copying Elements from one array to another



```
void copyArray(int v1[], int v2[], size_t size)
{
  for(auto i  = 0; i < size; ++i)
  {
      v2[i] = v1[i];
  }
  return;
}

int main()
{
  int v1[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  int v2[10];

  copyArray(v1, v2, 10);

  for (auto elem : v2)
      std::cout << elem << std::endl;
  return 0;
}
```

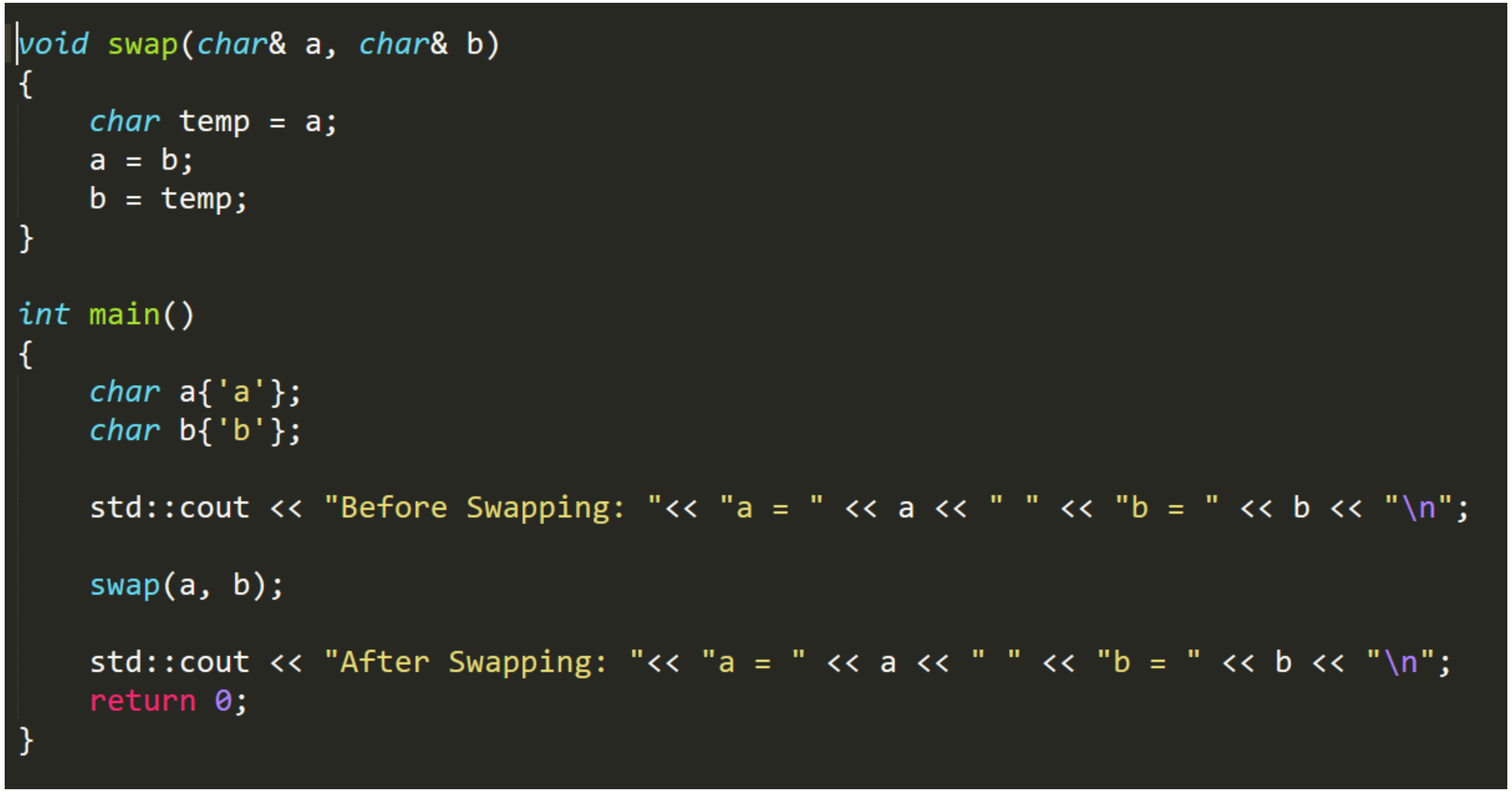
C++ Basics: Pointers (Examples)



```
int *intptr; // Declare a pointer that holds the memory address
             // of a variable of type int

intptr = new int; // Allocate memory from free store and store
                  // the address in intptr

*intptr = 100;   // Store 100 in the memory location pointed
                 // by intptr
int a = 5;

intptr = &a;
```

- Try to ensure that a pointer always points to a value, if you have a case where you have not yet assigned an object to point to use “nullptr”. nullptr is shared by all pointer types.

C++ Basics: Pointers(contd.)



```
void swap(char& a, char& b)
{
    char temp = a;
    a = b;
    b = temp;
}

int main()
{
    char a{'a'};
    char b{'b'};

    std::cout << "Before Swapping: "<< "a = " << a << " " << "b = " << b << "\n";

    swap(a, b);

    std::cout << "After Swapping: "<< "a = " << a << " " << "b = " << b << "\n";
    return 0;
}
```

- Passing parameters by references. Reference is like a pointer, except you don’t need * to deference it and it cannot refer to a different object after it has been initialized.

Tools of the trade

- Compilers: G++, Clang, MSVC++, ICC
- IDEs: Eclipse, VSCode, MS Visual Studio, Xcode…
- Source Control Management
- Debuggers: GDB, LLDB
- Style Guides
    - http://google-styleguide.googlecode.com/svn/trunk/cppguide.html
    - https://www.boost.org/development/requirements.html
- CPP Standards https://isocpp.org/
- References
    - C++ Programming Language, 4th Edition by Bjarne Stroustrup
    - https://en.cppreference.com

Session 1 Practice Exercises

- Debug DebugExercise.cpp

Next Session

- Procedural programming aspects of C. We will start writing simple functions, explore recursion and learn some STL containers like vector and map.
- System Setup Instruction:
  - The most effective way to learn programming is to follow along in these sessions with actual code.
  - For Basic cpp practice, you can use a web IDE cpp.sh
  - If you are already setup or prefer an IDE/Compiler, you can use that.
  - Else, we recommend using Code::Blocks which is supported across all major OSes.