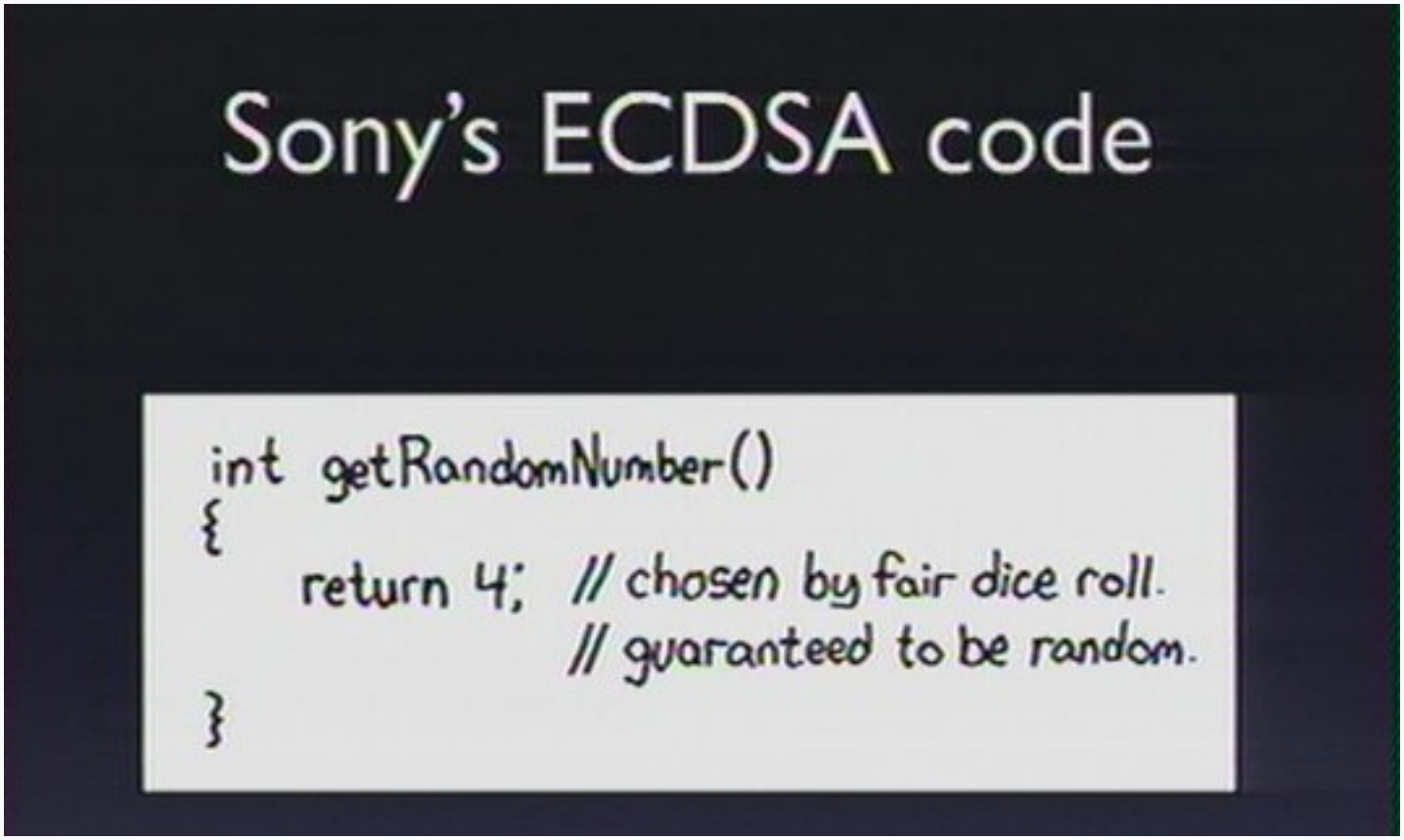
# C++

## SWE315

## Lesson 15

Prof. Zachi Baharav
zbaharav@cogswell.edu


Sony's ECDSA code
int getRandomNumber()
{
return 4; // chosen by fair dice roll.
// guaranteed to be random.
}

# Lesson 15

- Reminder (from Deep/Shallow copy) :
  - Implicit class members created by C++.
- Templates
  - Functions
  - Classes
  - Expression parameter
  - (not) Specialization
- Container classes (later on)
  - Lab: work
  - Good easy resource: www.learncpp.com

# Implicit member functions

- Default constructor
- Copy constructor ←
- Assignment operator ←
- Default destructor
- Address operator

# Copy .vs. Assignment

- Copy : Into NEW object.
- Assignment: Into existing object.

- There are three general cases where the copy constructor is called instead of the assignment operator:
  - When **instantiating one object and initializing it** with values from another object.
    - Point P1(0,0) ;
    - Point P2 = P1 ;  // <-- COPY constructor
  - When **passing an object by value**.
  - When an object is **returned from a function by value**.

# Summary (Copy/assignment)

- The default copy constructor and default assignment operators do shallow copies, which is fine for classes that contain no dynamically allocated variables.
- Classes with dynamically allocated variables need to have a copy constructor and assignment operator that do a deep copy.
- The assignment operator is usually implemented using the same code as the copy constructor, but it :
  1. checks for self-assignment,
  2. returns *this, and
  3. deallocates any previously allocated memory before deep copying.
- If you don't want a class to be copyable, use a private copy constructor and assignment operator prototype in the class header.

# Templates (functions)

- Function templates.

```
int max(int nX, int nY)
{
    return (nX > nY) ? nX : nY;
}
```

- Duplicate code is trouble.

```
double max(double dX, double dY)
{
    return (dX > dY) ? dX : dY;
}
```

- If you overload the '>' operator, than the same function can hold for everything (classes etc).

# Templates (functions)

- Can use ‘class’ in place of ‘typename’

```
template <typename T1> // this is the template parameter declaration
Type max(T1 tX, T1 tY)
{
    return (tX > tY) ? tX : tY;
}
```

```
template <typename T1, typename T2>
```

```
int nValue = max(3, 7); // returns 7
double dValue = max(6.34, 18.523); // returns 18.523
char chValue = max('a', '6'); // returns 'a'
```

# Template functions

- Process: Compiler creates a “function Template instance”

- Drawbacks:
  - Bizarre compilation error massages!
  - Older compilers have issues with.

# Template classes

- "Template classes are ideal for implementing container classes, because it is highly desirable to have containers work across a wide variety of data types, and templates allow you to do so without duplicating code. Although the syntax is ugly, and the error messages can be cryptic, template classes are truly one of C++'s best and most useful features."

- (verbatim from http://www.learncpp.com/cpp-tutorial/143-template-classes/ )

```
template <typename T>
class Array
{
private:
    int m_nLength;
    T *m_ptData;

Public:

Array(int nLength)
{
    m_ptData= new T[nLength];
    m_nLength = nLength;
}

T& operator[](int nIndex)
{
        return m_ptData[nIndex];
}
int GetLength(); // templated GetLength() function defined below
};

template <typename T>
int Array<T>::GetLength() { return m_nLength; }
```

```
int main()
{
    Array<int> anArray(12);
    Array<double> adArray(12);

    for (int nCount = 0; nCount < 12; nCount++)
    {
        anArray[nCount] = nCount;
        adArray[nCount] = nCount + 0.5;
    }

    for (int nCount = 11; nCount >= 0; nCount----;)
        std::cout << anArray[nCount] << "\t" << adArray[nCount] << std::endl;

    return 0;
}
```

# Template classes

- "Template classes are ideal for implementing container classes, because it is highly desirable to have containers work across a wide variety of data types, and templates allow you to do so without duplicating code. Although the syntax is ugly, and the error messages can be cryptic, template classes are truly one of C++'s best and most useful features."

- **Specialization of template classes.**
  - **For example, special printing for 'double' and 'int'.**
  - **When need deep-copy (memory assigned etc). For example, when the container class uses pointers to objects.**

- (verbatim from http://www.learncpp.com/cpp-tutorial/143-template-classes/ )

# END