Katerina Chinnappan

CMPS 109

Assignment 1

C++98/11/14/17 Features

C++ is a compiled programming language with a general-purpose which combines the properties of both high-level and low-level programming. Compared with its predecessor, the C programming language, the most attention goes to object-oriented and generic programming. The name <<C++ >> comes from the C programming language, where the <<++>> operator stands for an incremented value.

The C++ programming language was created in the early 1980's by Bjarne Stroustrup. Initially, the C++ programming language was developed to avoid programming in assembly language (machine language), and C. Its main purpose was to make the writing of programs easier and enjoyable for individual programmers. There was never a paper development plan for C++; design, documentation and implementation progressed at the same time. C++ was developed and is continued to develop in all directions to cope with the difficulties faced by the users.

The C++ language is widely used for software development. For example, variety of applications, development of operations systems, device drivers, video games and much more. There are several implementations of the C++ language – both free and commercial. Projects such as GNU, Microsoft produce them. The GNU Project is a project to develop open source software (OSS).

C++98, the first standard version of programming language C++ was developed by a working group ISO/IEC JTC1/SC22/WG21 and published in 1998. The standard consists of two parts – the basics of the language (core language) and the standard library, which includes the Standard Template library (STL) and a modified version of the C standard library.

Though C++98 was just the beginning of the standard version, it lacked things such as automatic type deduction, support for `nullptr` and most importantly "rvalue references — a feature that augurs a paradigm shift in how one conceives and handles objects" (http://blog.smartbear.com/). 13 years later since C++98, C++11 was approved by the C++ committee and published with many significant changes in the language. The major goals to improve C++11 was to support data abstraction, object-oriented and generic programming.

Some major features and improvements in C++11:

1) Automatic type deduction
Example:
```
auto numOfHolidays = 14; //numOfHolidays is of type int because 14 is a whole
//number.
```

2) `nullptr` keyword (can now replace macro NULL)

Example:

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

Node temp = head;
while (temp! = nullptr)
{
    temp = temp->next;
}
```

Multiple new features were added to STL for C++11 one of them is:

Additional smart ptr class . `shared_ptr && unique_ptr` both compatible with the STL components.

3) Class `shared_ptr` - is a handy tool that can solve many problems of the developer. `shared_ptr` implements the reference count. Contents will be freed when the reference count is equal to 0. It's an implementation of one of the basic rules of the garbage collector.

Example:

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

shared_ptr<int> orig(new int(100));
shared_ptr<int> temp(new int(50));

//After executing this line, the resource pointed earlier to
//temp (int (50)) is available and in the orig int (100) will refer to both
//pointers now
temp = orig;

//orig int (100) will be only available after the delete of the last referencing
//pointer to it
```

4) Class `unique_ptr` - this pointer replaces the old and problematic `auto_ptr`. The main problem lies in the ownership rights. The object of this class loses ownership rights while performing copy (assignment, using the copy constructor, transfer to the function by value). Unlike `auto_ptr, unique_ptr` prohibits copying.

Example:

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

unique_ptr<int> orig(new int(100));
unique_ptr<int> temp;

temp = orig; //error when compiling
```

5) Inheriting constructors. For example, when in the parent class (base) a constructor is declared, there can be several inheriting constructors declared in the child classes (derived). Example:

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

class Apple{
         Apple(string foo)
         {
              //do something
         };
};
class Seed : Apple{ //child of Apple
         Seed(string foo, int bar) : Apple(foo)
         {
              //inherited parent constructor
              //do something
         }
};
```

C++98 libraries have been improved in C++11 with the use of new language features. Two examples would be `noexcept` and `constexpr`.

6) With the new C ++ standard, more interesting and useful improvements have evolved, one of which specifies compile time `noexcept`, which tells the compiler that the function does not throw exceptions.
Example:

```
void no_exception () noexcept;//this functions will not throw exception
```

7) One of the new features C ++ 11 – is the specifier `constexpr`. With it you can create variables, functions, and even objects, which will be calculated at compile time. This is convenient, because previously we had to use templates for such purposes. First, a few words on what in general is a specifier `constexpr`. As already mentioned, it is possible to performs some operations at compile time. Take a look at the following example:

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

constexpr double loan(double monthly_payment)
{
	return monthly_payment;
}

void foo()
{
	double annual_rate = x / 100;
	double months = years * 12;
	double y = (1 + (annual_rate / 12));
	double w = Math.pow(y, months);
	double z = annual_rate / 12;
	double monthly_payment = (z) * (loan) * (w) / (w - 1);;

	constexpr double pay_month = loan(monthly_payment);
}
```

Keyword `constexpr`, added to the C ++ 11, before a function signifies if the parameter values may count at compile time, then the return value must also be counted at compile time. If at least one parameter is not known at compile time, the function will be launched in the runtime (and a non-compile-time error will be displayed).

```
constexpr type = expression;
```

The key word in this case, means the creation of a constant. And the expression must be known at compile time.
Consider this example:

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

constexpr int newAverage(int oper1, int oper2)
{
	return (oper1 + oper2)/2;
}

int average(int oper1, int oper2)
{
	return (oper1 + oper2)/2;
}

int main()
{
	int num1 = 2;
	int num2 = 10;
	int average1 = average(num1, num2); // Correct
	constexpr int average2 = average(num1, num2); // Wrong - constexpr to int ,
//wrong
	int average3 = newAverage(num1, num2); // Correct - as long as product 3 is
//not constexpr and != product()
	constexpr int average3 = newAverage(num1, num2); // Correct - constexpr
//varible
	return 0;
}
```

The 7 features listed above are features in C++11. Now I will present new features established in C++14. In April, a meeting was held in Bristol by the C ++ committee, on which the first proposals to amend the new standard C ++ 14 were considered. The following are some new features in C++14:

1) Generalized initialization of captured variables lambdas with support of capture-in movement. In C ++ 11 lambdas do not support the capture-in movement (capture-by-move). To move an object inside a lambda function, it was necessary to write some wrapper, like the old `auto_ptr`. Instead, to add the possibility of explicit capture-in movement, it was decided to add the support for generalized initialization of captured variables. For example,

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

void lambda()
{
	int a = 5;
	int lambda = [&r = a]{ r += 5}
	             ();//invoke lambda
	cout<<"a: "<<a<<endl;
}

int main()
{
	lambda();
}
//Output
//10
```

2) User literals for standard library types. C ++ 11 introduces the concept of user-defined-literals. The following were added to the standard user-defined-literals:

- The operator `s` for `basic_string`
- Operators `h, min, s, ms, us, ns` for the types in the chrono:: duration

Example:

```
auto name = "Katerina"s; // type std::string
auto time = 15min; // type chrono::minutes
```

3) `optional` keyword: This feature introduces a new type in the standard library, meaning an optional object. Possible ways to use:

- The ability to specify which parameters in the function are optional
- Ability to use null (without the use of conventional (the raw) pointers)
- Ability to manually control the lifetime of objects RAII (*Resource Acquisition Is Initialization)*

Example:

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

template<typename Name, typename String>
class Info
{
	optional<String> getName(Name name);
	//if class Info doesn't contain a particular
	//name, then nothing gets returned
}
```

4) `shared_mutex && shared_lock`

When developing multi-threaded programs, there is sometimes a need to give to an object multiple permissions to read/write. This feature adds to the standard `shared_mutex` library designed for this purpose. The `lock` feature enables a unique access, and can be used with the older `lock_guard` and `unique_lock` features. To have a mutual access, you must use `lock_shared` feature, and its better do it using the RAII class `shared_lock`:

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

void safe()
{
	std::shared_lock<std::shared_mutex> access(both_);
	//shared access, multiple have access
}

void unique()
{
	std::unique_lock<std::shared_mutex> access(both_);
	//one writer/thread has the access
}
```

5) `constexpr` variable templates: in C++11, C++14 allows us to use `constexpr` variables and make them templated. The benefit is that we can use the type of any variable in a generic function. Consider this example:

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

using namespace std;

template<typename Square>
constexpr Square side = Square(5);
//constexpr within template

template <typename Square>
Square area(Square side)
{
        return side<Square> * 4;
}
```

C++ continues to develop in various directions and this year in 2017, C++17 will get published soon. On March 5, 2016 in the city of Jacksonville the ISO C ++ committee of Congress took place. People gathered to decide what new features will be included in the standard C++17. C ++ 17 promised a major release, so we'll see. Let's look what will be included in C++17:

1) `string_view` – you can only view, but not own/copy. It will look good in the interface, but won't give the fundamental performance as compared with `optional`(C++11)
2) Lambda Capture of `*this` by Value as `[=,*this]`
   Expected feature. Passing a pointer to `this` was possible before, but it wasn't exactly what was needed. Now it is possible to do both ways.
3) Special mathematical functions - This is an expected feature in C++17. Spherical hypergeometric functions will now be part of C++17.
4) `constexpr` Lambda. It is not so much a desired feature, but bringing the language to a consistent state. We have lambdas and `constexpr`-function, well, apparently, should be lambda `constexpr` functions.

Now let's look at a feature that won't be included in C++17:

- A Module System for C++. Excellent feature! Modules have existed for a while and implemented in Clang and there's a new implementation from Microsoft. Of course, they are not compatible with each other (who would doubt), and the committee hasn't yet decided which one is better. Both approaches have some advantages and disadvantages, so deciding which one to include is a hard decision

Bibliography

"Std::literals::string_literals::operator""s." *Std::literals::string_literals::operator""s - Cppreference.com*. N.p., n.d. Web. 14 Jan. 2017.

"Numerical Aspects of Special Functions." *Special Functions* (2011): 333-48. Web.

Wording for Constexpr Lambda (n.d.): n. pag. Web.

"Capturing *this." *Capturing *this*. N.p., n.d. Web. 14 Jan. 2017.

"Noexcept Specifier (since C++11)." *Noexcept Specifier (since C++11) - Cppreference.com*. N.p., n.d. Web. 14 Jan. 2017.

Number:, Document, Date:, Working Group:, and Reply To:. "A Module System for C++." (n.d.): n. pag. Web.

"The Biggest Changes in C++11 (and Why You Should Care)." *Software Quality Matters Blog*. N.p., n.d. Web. 14 Jan. 2017.

Bancila, Marius. "Ten C++11 Features Every C++ Developer Should Use." *CodeProject*. N.p., 03 Apr. 2013. Web. 15 Jan. 2017.

2014, Mark Nelson September 16. "The C++14 Standard: What You Need to Know." *Dr. Dobb's*. N.p., n.d. Web. 18 Jan. 2017.

"An Overview of C++14 Language Features." *An Overview of C++14 Language Features*. N.p., n.d. Web. 18 Jan. 2017.

"C++14 Is Here: Summary of New Features." *InfoQ*. N.p., n.d. Web. 18 Jan. 2017.

"Final Features of C++17." *Final Features of C++17 - Meeting C++*. N.p., n.d. Web. 18 Jan. 2017.

"C++17 Feature List Is Now Complete, Enters Review." *InfoQ*. N.p., n.d. Web. 18 Jan. 2017.

LoopPerfect. "C++ 17 vs. C++ 14 - If-constexpr." *Medium*. N.p., 28 June 2016. Web. 18 Jan. 2017.