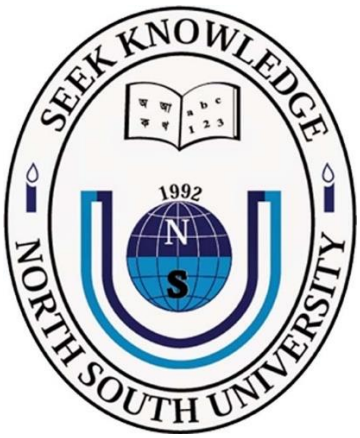

North South University

Slecture 02

Group: Group 25

Course: CSE 425 – Concepts of Programming Languages

Section: 3

Faculty Initial: MSK1

Faculty Name: Md. Shahriar Karim

Semester: Summer 2019

Date of Submission: 09th July 2019

Group Members:

1. **Humayara Binte Omar 1620484042**
2. **Fatema Tuz Zohra 1620191042**
3. **Nabiul Hoque Khandakar 1631164642**

What is scope?

The scope of a variable is the range in which it is visible. A variable is visible in a statement if it can be referenced in that statement.

Example of scoping:

```
int x = 4;              /* variable x defined with file scope */
2   long myfunc(int x, long y); /* variable x has function       */
3                               /* prototype scope               */
4   int main(void)
5   {
6      /* . . . */
7   }
```

Two types of scoping: Static scoping and dynamic scoping

Static scoping:

Lexical scoping (sometimes known as static scoping) is a convention used that sets the scope of a variable in such a way that it can only be called from within the block of code in which it is defined. The scope is determined when the code is compiled. A variable declared in this fashion is sometimes called a private variable.

Example: Languages such as C and Pascal apply to static scoping.

Advantages:

1. Readability
2. Locality of reasoning
3. Less run-time overhead

Disadvantages:

Some loss of flexibility

There are two categories of static-scoped languages:

1. In which subprograms can be nested, which creates nested static scopes. Nested scopes are created only by nested class definitions and blocks. Ada, JavaScript, Common LISP, Scheme, Fortran 2003+, F#, and Python allows nested subprograms but C do not.
2. In which subprograms cannot be nested. In this category static scopes are also created by subprograms but nested scopes are created only by nested class definitions and blocks.

Dynamic Scoping:

Dynamic scoping creates variables that can be called from outside the block of code in which they are defined. A variable declared in this fashion is sometimes called a public variable. Dynamic scoping is best avoided since it makes the code hard to read/ understand at first glance. Dynamic scope can be determined only at run time.

Advantages:

The only advantage of dynamic scoping is writability. It is more convenient and it provides more flexibility than static scoping. For example, in some cases, some parameters passed from one subprogram to another are variables that are defined in the caller. None of these need to be passed in a dynamic scoped language, because they are implicitly visible in the called subprogram.

Disadvantages:

1. The inability to statically check for references to non-local variables.
2. Dynamic scoping also makes program difficult to read because the calling sequence of subprograms must be known to determine the meaning of references to non-local variables.
3. There is no way to protect local variables from being accessed to by subprograms because local variables of subprogram are all visible to all other executing subprograms.
4. Accessing to non-local variables in dynamic scoping takes far longer than accesses to non-local variables when static scoping is used.

Static Scoping:

Example:

```
intx = 10;

// Called by g()
intf()
{
   returnx;
}

// g() has its own variable
// named as x and calls f()
intg()
{
   intx = 20;
   returnf();
}

intmain()
{
  printf("%d", g());
  printf("\n");
  return0;
}
```

OUTPUT: 10

To sum up in static scoping the compiler first searches in the current block, then in the surrounding blocks successively and finally in the global variables.

Dynamic Scoping:

Example:

```
intx = 10;

// Called by g()
intf()
{
   returnx;
}

// g() has its own variable
// named as x and calls f()
intg()
{
   intx = 20;
   returnf();
}

main()
{
  printf(g());
}
```

OUTPUT: 20

in dynamic scoping the compiler first searches the current block and then successively all the calling functions.

In static scoping a variable always refers to its top level environment.	In dynamic scoping, a global identifier refers to the identifier associated with the most recent environment.
In most of the programming languages including C, C++ and Java, variables are always statically (or lexically) scoped.	Dynamic Scoping is uncommon in modern languages.
Scope rules determined at compile-time.	Scope rules must be determined during run-time

Evolution of the Major Programming Languages:

Languages :	Reasons for success :	Comparison:
Plankalkül	• Advanced features in the area of data structure. • Inclusion of mathematical programs.	• Arrays and records in this language is called structs in C-based programming languages • It included iterative statement similar to Ada. • Similar assertion function like java.
FORTRAN(Formula Translation computing programming language)	• FORTRAN is the dominating language in the supercomputers. • FORTRAN compilers are mostly available. • It is earlier first programming language and the best at numerical analysis and technical calculations. • This language is reliable to the Meteorological Department involves extensive numerical weather computations. • A lot of existing code.	• FORTRAN has efficient compilers compared to the C++. • FORTRAN is the best for numerical computations compared to all other languages.
LISP (List Processing)	• Provide advanced features for list processing,	• LISP is different from the imperative languages,

	especially useful for artificial intelligence (AI). • Possible to write Lisp programs that rewrite other Lisp programs in arbitrary ways. • LISP functions can be manipulated in the same the way data is manipulated.	because it is a functional programming language and because the appearance of LISP programs is very different from those in languages such as , Java or C++.
ALGOL 60	• The concept of block structure was introduced. • Two different means of passing parameters to subprograms were allowed: pass by value and pass by name. • Procedures were allowed to be recursive. The ALGOL 58 description was unclear on this issue. Note that although this recursion was new for the imperative languages, LISP had already provided recursive functions in 1959. • Stack-dynamic arrays were allowed	• The biggest advantage of ALGOL over preceding languages such as FORTRAN and COBOL is that it encourages the production of well-structured programs.
COBOL(Common Business Oriented Language)	• It's easy to read. Its high-level English-like syntax can resemble a well-structured novel with appendices, cross-reference tables, chapters, footnotes and paragraphs. • It is self-documenting and appeals to proponents of readability. • It can handle huge processing volumes with ease. • It's still widely used for business applications, which is an area it excels	

	at. COBOL is relatively easy to develop, use, and maintain.	
BASIC (Beginner's All-purpose Symbolic Instruction Code)	• Simple language. • Its features made it popular o minicomputers. • Easy to learn.	• Much of the design of BASIC came from Fortran, with some minor influence from the syntax of ALGOL 60
PL/I (Programming language 1)	• Programs were allowed to create concurrently executing subprograms. • It was possible to detect and handle 23 different types of exceptions, or run-time errors. • Subprograms were allowed to be used recursively, but the capability could be disabled, allowing more efficient linkage for non - recursive subprograms. • Pointers were included as a data type. • Cross-sections of arrays could be referenced. For example, the third row of a matrix could be referenced as if it were a single-dimensioned array.	• PL/I is that it included what were then considered the best parts of ALGOL 60 - recursion and block structure ,Fortran IV (separate compilation with communication through global data), and COBOL 60 (data structures, input/output, and report-generating facilities), along with an extensive collection of new constructs, all somehow cobbled together
APL and SNOBOL	• Dynamic typing • Dynamic storage allocation.	
C	• As a middle level language, C combines the features of both high level and low level languages. It can be used for low-level programming. • C is a structured programming language which allows a complex program to be broken into simpler programs called functions. It also allows free movement of data	• The original C programming language is not object-oriented, which is the most significant difference between the two. C is what's called a “procedural” programming language, while C++ is a hybrid language that's a combination of procedural and object-oriented.

	across these functions. • C is highly portable and is used for scripting system applications which form a major part of Windows, UNIX and Linux operating system. • C is a general purpose programming language and can efficiently work on enterprise applications, games, graphics, and applications requiring calculations. • C language has a rich library which provides a number of built-in functions. It also offers dynamic memory allocation.	
Ada	• strong typing • Modularity mechanisms (packages). • Run-time checking • Parallel processing (tasks, synchronous message passing, protected objects, and nondeterministic select statements), • Exception handling, and generics.	
Smalltalk	• Smalltalk was the first programming language that fully supported object oriented programming.	• It was the first language to adopt object oriented programming, which absent in all the other languages before this.
C++	• C++ is a highly portable language and is often the language of choice for multi-device, multi-platform app development. • C++ is an object-oriented programming language and includes classes, inheritance, polymorphism, data abstraction and	• C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent. java is pure object oriented language but C++ is a both procedural and object oriented programming

	encapsulation. • C++ allows exception handling and function overloading, which are not possible in C. • C++ is a powerful, efficient and fast language. It finds a wide range of – from GUI applications to 3D graphics for games to real-time mathematical simulations.	language. Java and CPP both are object oriented programming languages.
Java	• Java was designed to be easy to use, write, compile, debug and learn. • Object oriented programming (OOP) is associated with concepts like class, object, inheritance, encapsulation, abstraction, polymorphism, etc. Java is object oriented programming language and allows you to develop modular programs and reusable code objects. • Java offers the comfort of write program once and run on any hardware and software platform and any Java compatible browser.	
Perl	• Simplified grammar • Lexical scoping • Object oriented programming • Extensibility.	
JavaScript	• No matter where you host JavaScript, execute always on client environment to save a bandwidth and make execution process fast. • JavaScript used to fill web page data dynamically such as drop-down list for a Country and State. • JavaScript syntaxes are easy and flexible for	

	developers. In-short easy language to get pick up in development. • The biggest advantages to a JavaScript having a ability to support all modern browser and produce the same result.	
PHP	• The high speed of PHP gives it an advantage over other scripting languages and provides functionalities such as the server administration and mail functionalities. • It is Open Source. Therefore, PHP is readily available and is entirely free.	• JavaScript is a client-side scripting language whereas PHP is a server-side scripting language.
Python	• Presence of third party Modules • Extensive support libraries • Open source and community development • Learning ease and support available • User-friendly data structures • Productivity and speed	• The biggest difference between the Java and Python languages is that Java is a statically typed and Python is a dynamically typed.

A good language design is quintessential in terms of efficiency and executability. Our early principles of a good design criteria only focused on efficiency in execution as machines were extremely slow and programming speed was a necessity. Earlier languages were designed to resemble the machine code to be generated to accelerate the execution time.

As we progress through time, efficiency on the user side has somewhat become the forefront of a well-designed programming language. Therefore, we have a new language evaluation criteria on our hands which cater to a programmer's efficiency, it is as follows:

1. Efficiency
   Design efficiency can be divided into 3 subgroups:

- Readability:
  How easy can the design in the programmer's head be mapped to code in that language and the ease with which programs can be read and understood.
- Writability:
  The ease with which programs can be developed for a given program domain. An expressive language allows for easy representation of complex processes and structures.

- Reliability
  The extent to which a program will perform according to its specifications or the dependability that the written program will work. Non-reliable programs tend to cost significantly at later stages.

2. Regularity
   Regularity ensures that there are no unusual restrictions, interactions, or behavior by assessing how effectively the features a language is integrated. In practice, there should be no surprises in the way the language features behave. Regularity can also be subdivided into 3 concepts:

   - Generality
     Generality rule advices to avoid special cases and generalizes related constructs into one construct. For example,
     In Pascal, it has 3 different loop structures (while, repeat, and for). Not a good example of generality
     In Ada, it has 1 loop structure with variations. A good example of generality

   - Orthogonality
     Orthogonality rule dictates that language constructs (i.e loops) should not behave differently in different contexts or there should be no unusual implications, should be meaningful. For example,
     In C, all parameters are passed by value except arrays, bad orthogonality

   - Uniformity
     Uniformity rule states that there should be a level of consistency between appearance and behavior of language constructs. For example, in C++, the operators "&" (the bitwise and) and "&&" (the logical and) look almost indistinguishable but formulate very distinct results, bad uniformity

3. Expressiveness
   It is the ease with which a language can express complex structures, meaning, the ease to write understandable code and conciseness of complex processes. For example,

In C, the line of code for copying strings, “while (*s++ = *t++);” is very concise and expressive for the operation it’s performing

4. Extensibility
   A programming language should be open in a sense that it should give the programmer the freedom to add features like:
   - New Data Types
   - Libraries and functions to libraries
   - Keywords

   Such additions enhances the user’s overall experience of the language.
   Also enables growth for the language by the help of the community i.e Python, Java, C version releases.
   An example of extensibility would be,
   In C++, operator overloading is allowed. Good extensibility
   In Java, it is not. Bad extensibility

5. Restrictability :
   A well designed language should enable a programmer to write programs successfully irrespective of having vast knowledge of the language or its constructs. For example, designing a language with a similar syntax to a popular language allows users to pick up the language in a short period of time due to following common language conventions.

6. Security
   Programs being written in the language should not cause any errors which hamper reliability. Features like type checking ensures that the compiler doesn’t run into problems and secures the program to avoid any sort of damages.

Criteria	Definition	Example
Efficiency	The ease with which programs be written in the language	Python, a well designed language, that programmer’s head can be mapped to code easily according to public acceptance in contrast to C/C++ or Java/FORTRAN
Generality(Regularity)	Avoid special cases and generalizes related constructs into one construct	C lacks nested function definitions(Bad Generality)

Orthogonality(Regularity)	Language constructs should not behave differently in different contexts.	C passes all parameters by value, except arrays, which are passed by reference(Bad Orthogonality)
Uniformity(Regularity)	A level of consistency should be there between appearance and behavior of language constructs	In C++, the operators & (bitwise and), && (logical and) yield very different results, but look confusingly similar(Bad Uniformity)
Expressiveness	It is the ease with which a language can express complex structures	while (*s++ = *t++); in C. Very expressive, very concise(Good Expressiveness)
Extensibility	There should be some general mechanism by which the user can add features to a language.	Version releases on popular languages like python, java etc.
Restrictability	Language design should enable a programmer to program usefully having minimal knowledge of the language.	A language should maintain consistency with accepted notations and conventions
Security	Programs should not run into runtime or compile time errors or damages	Types, type checking, and variable declarations

References:

https://ivanapurnomo.wordpress.com/2013/04/08/concepts-of-programming-languages-chapter-5-names-bindings-and-scopes/

http://ngocchan-nguyen.blogspot.com/2011/03/what-are-advantages-and-disadvantages.html

http://evankristia.blogspot.com/2015/11/chapter-5-review-question-and-problem.html

https://www.geeksforgeeks.org/static-and-dynamic-scoping/

http://evankristia.blogspot.com/2015/11/chapter-5-review-question-and-problem.html

https://whatis.techtarget.com/definition/lexicalscopingstaticscoping

http://evankristia.blogspot.com/2015/11/chapter-5-review-question-and-problem.html

https://whatis.techtarget.com/definition/lexical-scoping-static-scoping

https://www.geeksforgeeks.org/static-and-dynamic-scoping/

www.cse.aucegypt.edu/~rafea/CSCE325/slides/2/LanguageDesign.pdf