Dear friends,

Third installment in your hands. We discuss the concept of name and the concept of variable at length. I look forward to your feedback.

Regards,

N S Kumar
ask.kumaradhara@gmail.com

Name:

What is in a name? “Rose by any other name smells as sweet”. There are so many entities which are part of a language and programs in these languages that we will have to give names to identify them. A name carries an abstraction with it which makes it easier to refer to it without describing the entity.

A language has to have a name. Some have been acronyms – sometimes invented with no reason or logic (what is 'BASIC'?). The name could be influenced by what the designer sees out of his window or what type of coffee he is enjoying at that point in time. The designer is influenced by some great personality – Scientist, Comedian, ... Some names are known for brevity(C, D, C++). You may wonder how your parents named you!

What are the various artifacts which have names in a language? Let us list a few.

- Variable
- constant
- function
- class
- object
- namespace
- package
- module

We shall try to understand about naming in general and then we shall delve on the concept of variable at length.

- Are the names case sensitive?
  - In some languages like C, C++, Java and Python, they are. In languages like Pascal, Cobol, Fortran they are not.
  - In the good old days, the keyboard (card punching machines) did not

support lowercase

- Is this good or bad?
  - Depends
  - we may use this as a convention – example : typenames and constants could be all in uppercase
  - most of the newer languages are all case sensitive

- What are reserved words and key words?
  - reserved word : specific meaning; cannot be changed
  - keyword : has specific meaning; can be changed
  - In some languages, all keywords are reserved : Cobol
    - making an indentifier is difficult; may result in compile time error
  - In some, no reserved words : PL/I
    - we can write code like

      if if = then then ...
  - In java, goto is a reserved word, but not a keyword – is not implemented
  - Most of them have a mix

  - What about introducing a new concept in a language?
    - overload existing keyword?
      - This affects the readability.
      - What does the word final mean in Java?
      - What does static mean across multiple languages?
    - add a new keyword?
      - Breaks existing code if the name has been already used.
    - contextual keyword in C++ 11/14
      - add a keyword at a position there would never have been an identifer.

- What are noise words?
  - Should we allow words which increase readability without affecting the semantics?
  - If <Expr> then
    - then may be dropped without affecting the semantics – 'then' is a noise word in Cobol.
  - Not commonly used in newer languages.

- What about the length of the name?
  - How long can the name be? What if the name should be used outside the program by the linker, operating system? What about the rules for making these names?
  - PL/I on IBM main frame limited the external name to 7 characters – would consider the first 4 and the last 3 characters.
  - In some languages, the name can be any length – but there should be some difference in the first few characters – could be 32 characters.
  -
- What are the naming syntax and conventions?
  - Special symbols in the name
    - first symbol(s)

      $ @ % in Perl

      @ @@ in Ruby

      r value usage in case of shell
    - underscore (_), hyphen(-) in the name
    - unicode characters
    - special symbols at either end - __somename__ as in Python
    - special symbols after the first characters
      - $? $$ as in Perl
    - space in the middle of a name

- What are naming conventions?
  - CamelCase
  - names separated by _
  - ___ at either end or both
  - all uppercase
  - name ending with _t

Variable:

The concept of variable is considered one of the most important aspects of programming. E. W. Dijkstra says this - “Once a person has understood the way variables are used in programming, he has understood the quintessence of programming”.

What are the different attributes of a variable?

1. Name
2. Location
3. Value
4. Type
5. Life
6. Scope
7. Storage Class
8. Qualifier

Let us discuss each one at length.

1. name:

Should every variable have a name?

What if we access through pointer a dynamic location?

int p = (int) malloc(sizeof(int));

*p is an int; but there is no name.

What about the compiler created temporary variable?

Assignment d = a + b + c; would become

t1 = a + b; d = t1 + c;

where t1 is a compiler created temporary for which we have no name in our vocabulary.

Can we make a variable on the fly during the program execution?

2. location:

This is influenced of Von Neumann architecture
(do not want call it address - is a low level
feature - confusing - is this physical or logical?
What if register - does not support & operator in 'C')
wrong in the book: address is not called the l-value

cannot do this:
int a = 10; &a = ...; // Error

can we assign to any entity having a location?
can a variable have more than one location?
can we find the location programmatically?
alias:

- multiple variable with the same address
- union
- through pointers
- through function calls
- through assignment
- while looping

You may want to create examples to illustrate these ideas.

In some languages, a value has a location and not the variable. It is possible for number of variables to share the same location. This is somewhat similar to ln command in unix. This gives rise a series of interesting questions?
What if the value of one of the variables is changed?
What if all the variables are changed? What happens to the value?

If the value is a composite(an aggregate), What happens if the variable is changed? What happens if the composite is changed through the variable?

Check: In Python,

```
a = 10; b = a ; print(id(a), id(b))
a = 20; print(b); # has b changed?
a  =  [10, 20];  b = a ; print(id(a), id(b))
 a.append(30); print(b)
a  =  [10, 20];  b = a ; print(id(a), id(b))
a = [30, 40]; print(b)
```

3. value:

Does a variable have a single value? Does it have components? Is this a value type or a reference type?

How do we initialize or assign to the variable?

When is the variable initialized?

Does the variable have a default value ?

If yes,

does it depend on the storage class?

does it depend on whether it is a local variable?

does it depend on the type?

If no,

What should happen when an uninitialized variable is used?

Does it give a compile time error

Does it give a runtime error

Is it an undefined behaviour

You may want to experiment these with C, Java and Python.

If the variable is an aggregate,

can we initialize partially?

If yes, can we decide which component to initialize?

What happens to other uninitialized components?

Would initializing a variable

cause call to a function?

create resources outside of the memory allocated?

Be compile decision or link time decision or load time decision or runtime decision?

If the variable is an aggregate,

can we assign one to another?

Would that result in aliasing (a shallow copy)?

Would that copy all the components(deep copy)?

To what level, is the copying done(recursive deep copy)?

4. type

The concept of type exists in Mathematics and also in the real world. A chair is a type. A wooden chair, a plastic chair, a steel chairs are all of the type chair. A type encapsulates common attributes and behaviours.

The concept of type in programming languages includes the following two aspects,

1. set of values
2. set of operations

A language cannot support all possible types. So, many languages provide a mechanism to make user defined data types.

Class : user defined type

Technically type is an interface (says what and not how)

A class is both a type as well as implementation of operations.

Should a variable have an explicit type?

How do we infer the type?

Is this based on the naming convention? Should the first letter or the symbol matter?

Do we declare a variable with respect to type?

Is the type known at compile time or is that determined at runtime?

If it to be known at compile time, can the compiler be asked to determine the type based on initialization?

Check: var in C# and auto in C++

if the type is a runtime mechanism, is the type determined based on the type of the expression assigned to it?

If the type is a runtime mechanism, can the type be changed at runtime? In that case, would the operator resolution happen at runtime? Would this be polymorphic?

If the values are stored as strings, how can we extract the value based on the operators?

Would these types form a hierarchy?

5. life

This is about existence of a variable – variable has some location in memory.

A few queries:

Can a variable exist at more than one location?

When does a variable get life? what does it depend on?

How long does it stay?

What about: register int a? // good, bad or ugly

Can a variable exist when the block in which it is defined is exited?

If yes, can these be referred later on by other routines?

If yes, when will this location be ultimately removed?

You may want to check the concepts like

reference counting

closure

decorator

6. scope

This is about the visibility of variables.

A few queries:

Can we have a variable with life and no scope?

How about scope without life?

Can we have names referring to different variables with overlapping scopes?

Can we access them using qualified names?

When is the scope decided? Compile time? Runtime?

We shall discuss more about life and scope later in this course.

7. qualifier

Can we change the attribute of a variable by prefixing a keyword?

How about these

long short unsigned ?

volatile

8. storage class

Can we change the way the variable is stored?

Where is it stored? How long?

These are a few keywords associated with storage class.

register auto extern static global local ...