Example work

Logical operators

and - &

Or - |

Not - !

Short-circuit ‘and’,’or’ - &&,||

Precedence: not->and->or (read left to right)

Short circuit can write over normal operator (higher precedence)

Keywords

break, continue, elif, else, if, constant, static, class, func, for, while, null, int, return, import, None, from

* Measures -> Derived
* pitch, duration and (pitch and integer should be same data type/is pitch an alias)
* Note, Sheet
* Anything non-zero or not null will be interpreted as true

Naming conventions

* All identifiers are case sensitive
* Identifiers can be used to refer to primitive and derived data types, functions, constants
* All identifiers must begin with a letter and only include letters, underscores and numbers
* Identifier have to start
  + with an uppercase letter for derived data types (Note, Measure, Sheet)
  + with a lowercase letter for primitive data types (pitch, int, duration)
  + with an underscore character for functions
* An identifier declared within an outer scope cannot be redeclared within an inner scope

wrong

pitch 3sample! = 5; (Wrong: Cannot include special characters in the name, and identifier cannot start with a number ( also ! refers to Not logical operator))

right

pitch sample3 = 5;

wrong

Note Sample\_note = (420, 69);

io.play[sample\_Note]; (Wrong: note won’t play as case sensitivity implies ‘Sample\_Note’ is a non-existent identifier)

right

Note Sample\_note = (420, 69);

io.play[Sample\_note];

(keyword declaration)

wrong

int pitch = 256; (Wrong: pitch is a keyword so cannot be an identifier)

right

int pitch1= 256;

wrong

Measure sample\_Measure1 = new Measure[6]; (Wrong: derived data type has to start with an uppercase letter)

right  
Measure Sample\_Measure1 = new Measure[6];

(example on last rule i.e. nested scoping)

right  
func \_construct\_fib[int length] returns[Measure M]{

Measure M = new Measure[length];

pitch p = 110;

for(int i = 0; i < length; i++){

Note New\_note = (p, 1);

M.add(New\_note,i);

p = p \* 2;

}

return [M];

}

wrong

func construct\_fib[int length] return(Measure M)

Reason: 3 mistakes-> name of the function, return to returns, round to square brackets

Criticisms

* Lack of flexibility (sparse data type support) done
* Cannot use the same variable names in different scopes as well(nahi this we have valid reason) but it may be useful in some cases, we’ll just mention with a valid reason (if we dont find anything else XD)
* No classes, just procedural done
* Though both Measure and Note are derived data types, Measure is declared with ‘new’ where as Note is not (confusing declaration) done
* Lack of shorthand operations

Criticisms

No language is perfect. Beats++ is no exception to the rule. Since it is built with a very specific use case in mind, a few compromises are made, some of which are listed below:

* Sparse data type support
  + Music oriented data types such as Note, Measure and Sheet are central to the working of programs. Rudimentary arithmetic data types play a supporting role in condition evaluation and iteration statements. Hence, there are only two arithmetic data types of fixed bit size- int and double. This is a flexibility constraint for the programmer and also a compromise on efficiency in cases where a smaller bit-width data type would be sufficient.
* Confused Declaration in few cases:
  + Declaration of a Note:

Note Note1 = (Note1\_pitch, Note1\_duration);

* + Declaration of a Measure:

Measure Measure1 = new Measure[Measure1\_Duration];

Both Note and Measure are derived data types in the language, but the declaration of one of them(Measure) is done using the keyword ‘new’ but this is not the case with the other one (Note)

* Purely procedural
  + Classes are omitted from Beats++ as there is very limited use for the same within the confines of the language. However, in the occasional program where a need to combine functions with a group of variables is felt, there will have to be a degree of redundancy admitted by way of writing the variable groups and functions separately.
  + In addition access control of functions to operate on specific variables will be difficult to achieve (here we are talking about variables with the same datatype, that would be indistinguishable to a function from an access standpoint)
* Lack of shorthand operations
  + The language does provide a nice way to represent music but it can get tedious in a few cases. Like if we want to declare an already existing note(meaning with the same pitch) but with a different duration, the whole note has to be declared again. There is no shorthand operation to copy the previous note and modify it.
  + Another place where it gets tedious is while adding notes to the measure. Each note has to be added individually, using the add operation, and direct shorthand concatenation is not possible.
* Lack of redeclaration support:
  + A decision made largely for increased readability, identifiers aren’t allowed a re-binding to a new variable on progression of program control to an inner scope. A minor hitch, but an inconvenience for programmers who are used to an allowance for the same.