|  |
| --- |
| Circle Language Spec |

## Summaries of Other Languages

This document contains summaries of other languages, mainly C and C++, in the form of bulleted lists. They were made around the year 2004 during an analysis of the grammar of C and C++ to compare them to the new computer language.

### C

*A summary of Msdn Library Visual Studio July 2000: C Language Reference*

* Identifiers
* Constants: floating point, integer, string, enumerator
* Lifetime: when does a variable exist
* Scope: visibility, reuse of names
* Name Spaces
* Variable
* Specifiers
  + Access modifiers: public, private,
  + Types: void, char, short, int, long, float, double, signed, unsigned, struct-or-union-specifier, enum-specifier, typedef-name; signed, unsigned
  + Type qualifiers: normal, const, volatile
* Declarations
  + Single value variable with fixed size declations
  + Array, string declations
  + Pointer declations
  + Enumeration declations
  + Structure declations
  + Union declations
  + Alignment
  + Typedef declations
  + complex variable declarations
  + initialization: scalar, aggregate and strings
  + Incomplete types (look it up)
* Statements
  + expression (can include assignment)
  + break
  + label, goto
  + while, do-while, for, continue
  + if, switch
  + null statement
  + return
  + try-except, try-finally
* Expressions and assignments
  + L-value and r-value expressions
  + Constant expressions
  + side effects
  + sequence points
  + operators (are functions)
    - Postfix operators
      * One dimensional array
      * Multidimensional array
      * Function call
      * Structure and union
      * increment, decrement
    - Unary operators
      * prefix increment, decrement
      * cast
      * sizeof expression
      * sizeof(type)
      * &, \*, +, -, ~. !: bitwise and, dereference, plus, negate, bitwise complement, not/logical negation
    - Arithmetics (are functions)
      * Multiplative: \*, / and %
      * Additive: +, -, pointer arithmetic
    - Bit shift
      * shift: <<, >>
    - Relational and equality / Comparison: <, >, <=, >=, ==, !=
    - Bitwise:
      * And: &
      * Xor: ^
      * Or: |
    - Boolean / logical operators:
      * And: &&
      * Or: ||
    - Ternary: Conditional expression: ? :
    - Assigment: =, \*=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=
    - Sequential evaluation: , (doesn’t do much)
  + Operator precedence
  + Type conversions
    - Assignment conversions: value of one type assigned to variable of another type
    - Cast conversions: cast of a value of one type to another type
    - Function call conversions: value of one type passed to a function taking another type
* Functions
  + declaration
  + return statement
  + definition
  + return type
  + parameters
  + arguments
  + variable number of arguments
  + prototypes
  + calls
    - call
    - arguments
    - call with variable number of arguments
    - recursive function

### C++

*A summary of Msdn Library Visual Studio July 2000: C++ Language Reference*

I reorganized the information into the following main topics:

* Identifiers
* Types
* Expressions
* Statements
* Declarations
* Types
* Scope

More detail:

* Identifiers
* Types
  + Simple
  + Derived
  + Nested types
* Statements
  + Statement types
    - Expressions
    - Declarations
    - Selections
    - Iterations
    - Jumps
    - Exception Handling
  + Blocks
  + Labeled statements
  + Compound statements
* Expressions
  + Primary expressions
  + Operators
    - Access

:: :: [ ] ( ) . –> , =

* + - Arithmetic

++ -- ++ -- + - \* / % + – \*= /= %= += –=

* + - Creation

new delete delete[ ]

* + - Pointer

\* & .\* –>\*

* + - Comparative

< > <= >= == !=

* + - Boolean

! && ||

* + - Bit

~ << >> & ^ | <<= >>= &= |= ^=

* + - Analytic

sizeof sizeof ( ) typeid( )

* + - Conversion

(type) const\_cast dynamic\_cast reinterpret\_cast static\_cast

* + - Conditional

e1?e2:e3

* + Semantics
* Declarations
  + Specifiers
  + Declarators
  + Functions
  + Enumerations
  + Templates
  + Namespaces
* Types
  + Types
  + Unions
  + Derived types
  + Objects
  + Overloading
* Scope

Full summary:

* Identifiers: (the distinction between identifiers and names is probably a flaw in the explanations)
  + Identifier:

**Name of** object, variable, type, structure, enumeration, union, member of type, member of structure, member of union, member of enumeration, function, function, typedef, label, macro, macro parameter

* + Name:

identifier, simple type, template, value

* Types:
  + Simple

void, unsigned, signed, short, long, int, float, bool, char, double

* + - Numerical limits
    - Standard conversions (see doc)
  + Derived:
    - Directly derived types

arrays, functions, pointers, object references, constants, type member pointers

* + - Composed/aggregate/elaborate types

type, structure, union, enum

* + Nested types: derived types declared inside a type
* Statements
  + Statement types
    - Expressions
    - Declarations
    - Selections: if, else, switch, case, default
    - Iterations: for, do, while
    - Jumps: break, continue, return, goto
    - Exception Handling: try, throw, finally, except, catch,
  + Blocks
  + Labeled statements
  + Compound statement: in case of if’s and switches and loops
* Expressions: compute, assign, generate side effects
  + Primary expressions
    - Literals: integer, character, floating point, string
    - Identifier
    - Operator (-function name)
    - Parenthesis
    - this
  + Operators
    - Access

:: :: [ ] ( ) . –> , =

Scope resolution, Global, Array subscript, Function call, Member selection (object), Member selection (pointer), Comma, Assignment

Comma : argument seperation, multiple statements where one expected (e.g. if specification)

* + - Arithmetic

++ -- ++ -- + - \* / % + – \*= /= %= += –=

Postfix increment, Postfix decrement, Prefix increment, Prefix decrement, Unary plus, Arithmetic negation (unary), Multiplication, Division, Remainder (modulus), Addition, Subtraction, Multiplication assignment, Division assignment, Modulus assignment, Addition assignment, Subtraction assignment

* + - Creation

new delete delete[ ]

Allocate object, Deallocate object, Deallocate object

* + - Pointer

\* & .\* –>\*

Dereference, Address-of, Apply pointer to type member (objects), Dereference pointer to type member

* + - Comparative

< > <= >= == !=

Less than, Greater than, Less than or equal to, Greater than or equal to, Equality, Inequality

* + - Boolean

! && ||

Logical NOT, Logical AND, Logical OR

* + - Bit

~ << >> & ^ | <<= >>= &= |= ^=

Bitwise complement, Left shift, Right shift, Bitwise AND, Bitwise exclusive OR, Bitwise OR, Left-shift assignment, Right-shift assignment, Bitwise AND assignment, Bitwise inclusive OR assignment, Bitwise exclusive OR assignment

* + - Analytic

sizeof sizeof ( ) typeid( )

Size of object, Size of type, type name

* + - Conversion

(type) const\_cast dynamic\_cast reinterpret\_cast static\_cast

* + - Conditional

e1?e2:e3

* + Semantics
    - Evaluation order and sequence points
    - Declarations on first use (in the middle of a statement)
* Declarations: Introduce names into a program
  + Specifiers: specify mainly type, access, storage. Characteristics of what is declared.
    - All:

public, protected, private, static, auto, register, extern, virtual, inline, const, volatile, friend, type, typedef

* + - Access:

public, protected, private

* + - Storage type:

static, auto, register; extern (linkage); Ommitted

* + - Function:

virtual, inline

* + - Type qualifiers:

const, volatile

* + - Friend: a global function overloaded and specified inside a type for access to its protected and private members

friend

* + - Type:

any type

* + - typedef specifier:

typedef

* + Declarators: specifies
    - The name and pointering in a declaration (and type names in certain cases). Basically the part of the declaration that the specifiers do not cover.
      * Name
      * Pointers: pointer, reference, array, pointer to any member of a type (ClassName::\*)
      * Type names: in conversions, sizeof, new, function prototypes, typedefs
    - Initialization
      * Pointers to const objects
      * Static members
      * Aggegrates
      * Character arrays
      * References
  + Functions
    - Function declaration
      * Argument declaration list
        + in prototypes
        + in definitions
      * Default arguments
      * Variable argument lists
      * function overloading
      * Return restrictions: Can’t return arrays or functions. They can however return references of pointers to arrays or functions, or arrays inside structures
    - Function definition
      * Distinction of declaration and definition: only in functions (?and maybe types too?)
  + Enumerations: Declaration, Conversions and enumerated types
  + Templates: functions and types: template
    - No arbitrary textual substitution, but only substitution of type, type or constant
      * type placeholders: <type …>
      * constant placeholder: any type name e.g. <int …>
      * type placeholder: <typename …>
    - Specifies a family of types, code generated when it is referenced in code
  + Namespaces
    - Declaration
    - Definition (like type)
    - Members
    - Namespace alias: alternate name for namespace
    - namespace
    - using namespace <namespace>: allows you to ommit the namespace identifier when referencing things in the namespace
    - Accessing the namespace (C++ uses ::)
  + Abstract declarator: identifier omitted
  + const and volatile pointers
* Types
  + Types
    - Defining types
    - Type objects
    - Type name
    - Type members
      * Functions
      * Data
      * Nested types, structs and unions
      * Enums
      * Bit fields
      * Friends
    - this pointer
    - Static data members
  + Unions:
    - Can have functions, can have constructors and destructors, can’t have base types and cannot be used as base types
  + Derived types
    - Multiple base types
    - Virtual functions
    - Virtual base types
    - Overriding (more on that in another summary)
  + Objects: special functions
    - constructor, destructor
    - conversions
    - new operator
    - delete operator
    - assignment operator
  + Overloading
    - argument type differences
    - argument matching to find the right function
    - Overloaded operators: any operator
* Scope
  + Scope rules
  + Scope: local (within block), function scope, file scope, type scope, prototype scope (check what they meant by that)
  + Lifetime, point of initializing objects.
  + Hiding names: one name takes precedence over another. Accessing a hidden name.
  + Name ambiguity
* Other aspects
  + Startup and termination:
    - exit abort, return
    - Command line
    - atexit function
  + Comments
  + Calling conventions
  + asm

#### Unclassified

* Microsoft specific

allocate3 \_\_inline property

\_\_asm1 \_\_int8 selectany

\_\_based2 \_\_int16 \_\_single\_inheritance

\_\_cdecl \_\_int32 \_\_stdcall

\_\_declspec \_\_int64 thread

dllexport3 \_\_leave \_\_try

dllimport3 \_\_multiple\_inheritance uuid

\_\_except naked3 \_\_uuidof

\_\_fastcall nothrow3 \_\_virtual\_inheritance

\_\_finally

* Keywords:
  + Constants: true, false
  + Don't know: typename, typeid, type\_info, mutable, explicit, bad\_cast, bad\_typeid

Where did the embedded declarations in a function declaration go?

Other Software System components replace functionality of Standard libraries, but C programmers should be able to specify functions from them, shouldn’t they. Not gonna mind right now.

Formal arguments are the variables inside the function as opposed to the values given in the function call.

Concepts

### Other

I don’t summarize Com, because it doesn’t really add any programming concepts, but mainly only libraries. I don’t summarize Visual Basic, because it doesn’t really add any concepts to… oh, I do need Visual Basic and Com, for the events and stuff… shit. And properties, and all. Many Visual Basic and Com concepts have no place in this programming language, but in other modules.

Com concepts: components, interfaces.