Glossary

activation record Data structure created for each procedure call. It contains parameters, return address, return result address, local variables, and temporary storage.

alias When two or more variables share the same location or two or more pointers point to the same memory cell, they are aliases of each other. Essentially, it is one variable (pointer) but it is known under different names.

alpha conversion Renaming of bound variables in a lambda term. For example, lambda $x.x + z$ can be alpha converted to lambda $y.y + z$.

beta conversion (lambda $x.M)N = [N/x]M$. Substitute N for every free occurrence of x in M, renaming bound variables of M as needed to avoid variable capture.

(beta) reduction Modeling of program execution as directed by beta conversion: (lambda $x.M)N \rightarrow [N/x]M$.

bound and free variables A bound variable is simply a place holder. The particular name of a bound variable is unimportant. For example, the functions lambda $x.x$ and lambda $y.y$ define exactly the same function (the identity function). In contrast, the name of a free variable is important. Free variables cannot be renamed without potentially changing the value of an expression. For example, in the following code fragment (in which x appears as a free variable), we cannot rename x to y without changing the value of the expression: $3 + x$ (assume x is bound to 2 and y to 3).

class A class defines the behavior of its instances. Usually the class contains definitions of all methods shared by its instances.

class interface The interface of an object in Smalltalk is the set of messages that can be sent to the object without receiving the error "message not understood." The interface of an object is determined by its class, as a class lists the messages that each object will answer.

class template In C++, templates are the mechanism for parameterizing a class with a type or a function. This allows the programmer to implement a class that operates on values of an arbitrary type (e.g., a general-purpose array class that may store elements of any type).

class variable Class variables are shared by all instances of a class.

confluence Property of lambda calculus guaranteeing that, if an expression has a normal form, the normal form is unique. This implies that the order in which reduction rules are applied is unimportant.

conformance A relation between two types that serves as the basis of subtyping. Conformance relies on messages understood (i.e., the object's interface), not the internal representation or inheritance hierarchy.

constructor (in object-oriented programming sense) A procedure that is used for creating and initializing objects. In C++, a constructor is called after the object has been created and is used mainly to initialize the object's internal state.

dangling pointer A pointer referring to an area of memory that has been deallocated. Dereferencing such a pointer usually produces garbage.

data type induction A process of formal reasoning about properties of abstract data types.

denotational semantics A technique of describing the "meaning" of programs as mathematical functions, allowing people to prove theorems and reason about programs as mathematical entities.

dynamic lookup When a message is sent to an object at run time, dynamic method lookup is performed to determine which method should be called.

dynamic scoping The variable that is not defined in the current scope is looked for in the most recent activation record.

dynamic type checking Checking for type errors when the program is executed.

encapsulation Language mechanism for restricting access to some of the object's components.

exception Exceptions are a control transfer mechanism, usually used to treat a special case or handle an error condition.

exception handler When an exception-raising condition occurs, control is transferred to a special procedure called the exception handler.

fixed point A fixed point of function `f` is a value `x` such that `x = f(x)`.

funarg problem The failure of traditional stack-based implementations of procedure calls in the presence of first-class functions (functions that can be passed as procedure parameters and returned as procedure results).

upward funarg problem: The problem of returning a function as a procedure result; requires allocating activation records on the heap and returning a closure containing a pointer to code and a pointer to the enclosing activation record.

downward funarg problem: The problem of passing a function as a procedure parameter; requires a tree structure for activation records.

garbage Memory allocated by a program that will not be accessed in any future execution of a given program.

garbage collection An automatic process that attempts to free memory that is storing garbage.

halting problem The problem of determining whether a given program halts when executed on a given input.

higher-order function Function that takes other functions as arguments or returns a function as its result.

implementation of an abstract type Hidden internal representation of an abstract type.

inheritance An object's definition may be given as an incremental modification to existing object definitions, in which case it is said that the object *inherits* from other objects.

instance variable Local variable defined in each instance of the class. Instance variables of different instances are independent.

interface of an abstract type Operations visible to the clients of an abstract type.

lambda calculus Mathematical system for defining functions, proving equations between expressions, and calculating values of expressions.

lazy function A lazy function evaluates its arguments only when it needs them. For example, a lazy implementation of OR will not evaluate its second argument if the first argument is true, and thus the result can be determined without looking at the second argument. Compare with *strict* function.

L-value The L-value of variable x is the storage associated with x.

mark-and-sweep garbage collection A form of garbage collection that uses two phases: First, it marks all memory that can be possibly reached by the program from its current state; second, it frees (sweeps) all memory that has not been marked.

message A name and a list of parameter values. Objects in Smalltalk communicate by sending messages to each other.

method Code found in a class for responding to a message.

method dictionary Method dictionary of a class contains all methods defined in the class.

multimethods In a language that uses multiple dispatch, more than one argument to a message may be used to determine which method is called at run time.

normal form A lambda term that cannot be reduced.

objects Run-time entities that contain both data and operations.

overloading (ad hoc polymorphism) A function name is overloaded if it has two or more implementations associated with it. Different implementations are distinguished by type (e.g., function name "+" may have two implementations, one for adding integers, the other for adding real numbers).

parametric polymorphism A function is parametrically polymorphic if it has one implementation that can be applied to arguments of different types.

pass-by-reference Method of parameter passing: passes the L-value (address) of an actual parameter. Allows changing the actual parameter.

pass-by-value Method of parameter passing: passes the R-value (contents of address) of an actual parameter. Does not allow changing the actual parameter.

private data Private data can be accessed only by the methods of the class in which they are defined.

protected data Protected data can be accessed by the methods of the class in which they are defined as well as by the subclasses.

public data Public data can be accessed by the methods of the class in which they are defined, subclasses, and the clients.

raising an exception Raising an exception transfers control to an exception handler. Exception can be raised either implicitly if a certain condition occurs or explicitly by executing an appropriate command.

representation independence Property of a data type according to which different computer representations of values of the data type do not affect behavior of any program that uses them (i.e., different underlying representations are indistinguishable by the type's clients).

R-value The R-value of variable x is the contents of storage associated with x.

selector Name of a message (Smalltalk terminology).

static scoping The variable that is not defined in the current scope is looked for in the closest lexically enclosing scope.

strict function A function is strict if it always evaluates all of its arguments. Compare with lazy function.

subclass (derived class) If class `A` inherits from class `B`, `A` is the subclass of `B`.

subtyping Type `A` is a subtype of type `B` if any context expecting an expression of type `B` may take an expression of type `A` without introducing a type error.

superclass (base class) If class `A` inherits from class `B`, `B` is the superclass of `A`.

static type checking Checking for type errors at compile time.

tail recursion A function is tail recursive if it either returns a value without making a recursive call or if it returns directly the result of a recursive call. All possible branches of the function must satisfy one of these conditions.

template (in Smalltalk) Part of the object that stores pattern of instance variables.

type A basic notion (like set in set theory). A type can also be described as a set of values defined by a type expression. The precise meaning of a type is provided by the type system, which includes type expressions, value expressions, rules for assigning types to expressions, and equations or evaluation rules for value expressions.

type error A situation in which an execution of the program is not faithful to the intended semantics, i.e., in which the program interprets data in ways other than how they were intended to be used (e.g., machine representation of a floating-point number interpreted as an integer).

type inference Determining the type of an expression based on the known types of some of its subexpressions.

type tag A tag attached to each value and containing information about its type. Used for dynamic type checking.

variable capture This term is best explained by example. When an expression e is substituted for a variable x in another expression lambda $y.e$', without any renaming of variables, then free occurrences of y within e are "captured" (i.e., bound) by the binding lambda y. For example, in

$$(\lambda x.(\lambda y.x))y = [y/x](\lambda y.x)! = \lambda y.y,$$

y should be free, but accidentally becomes bound. To avoid variable capture, all bound variables should be renamed so that their names are different from those of all free variables and all other bound variables.

von Neumann bottleneck Backus' term for the connection between CPU and main memory. Every computer program operating on the contents of main memory must send pieces of data back and forth through this connection, thus making it a bottleneck.

vtable In C++, vtable is the virtual method table. It contains pointers to all virtual member functions defined in the class.

the Y combinator When applied to a function, it returns its fixed point. $Y = \lambda f.\ (\lambda x. f(xx))\ (\lambda x. f(xx))$.