AspectC++ Quick Reference

Concepts
Aspects are modular implementations of crosscutting concerns. They can affect join points in the component code, e.g. class definitions, or in the dynamic control flow, e.g. function calls, by advice. A set of related join points is called pointcut and defined by a pointcut expression.

Aspects
Aspects extend the concept of C++ classes. They may define ordinary class members as well as advice.

aspect A : public B { ... ;
defines the aspect A, which inherits from class or aspect B

Slices
A slice is a fragment of a C++ element like a class. It may be used by introduction advice to implemented static extensions of the program.

slice class ASlice { ... void f(); ... };
defines a class slice called ASlice
slice void ASlice::f() { ... }defines a non-inline member function f() of slice ASlice

Advice
An advice declaration specifies how an aspect affects a set of join points.

advice pointcut : around(...) {...}
the advice code is executed in place of the join points
advice pointcut : before/after(...) {...}
the advice code is executed before/after the join points in the pointcut
advice pointcut : order(high, ...low);
(high and low are pointcuts, which describe sets of aspects. Aspects on the left side of the argument list always have a higher precedence than aspects on the right hand side at the join points, where the order declaration is applied.
advice pointcut : slice class : public Base {...}
introduces a new base class Base and members into the target classes matched by pointcut.
advice pointcut : slice ASlice ;
introduces the slice ASlice into the target classes matched by pointcut.

Match Expressions

Match expressions are primitive pointcut expressions. They filter program entities based on their signature.

Type Matching
"int" matches the C++ built-in scalar type int
"%" matches any pointer type

Namespace and Class Matching
"Chain" matches the class, struct or union Chain
"Memory%" matches any class, struct or union whose name starts with “Memory”

Function Matching
void reset() matches the function reset having no parameters and returning void
"% printf(...)" matches the function printf having any number of parameters and returning any type
"% ...:%(...)" matches any function, operator function, or type conversion function (in any class or namespace)
"% ...:Service:%(...) const" matches any const member-function of the class Service defined in any scope
"% ...:operator %(...)"
matches any type conversion function
"virtual % C:%(...)" matches any virtual member function of C
"static % ...:%(...)" matches any static member or non-member function

Variable Matching
"int counter"
matches the variable counter of type int
"%guard"
matches the global variable guard of any type
"% ...:="
matches any variable (in any class or namespace)
"static % ...:="
matches any static member or non-member variable

Template Matching
"std::set<...>" matches all template instances of the class std::set
"std::set<int>" matches only the template instance std::set<int>
"% ...:=%(...)" matches any member function from any template class instance in any scope

Predefined Pointcut Functions
Predefined pointcut functions are used to filter, map, join, or intersect pointcuts.

Functions / Variables

call(pointcut) provides all join points where a named and user provided entity in the pointcut is called.
builtin(pointcut) provides all join points where a named built-in operator in the pointcut is called.
execution(pointcut) provides all join points referring to the implementation of a named entity in the pointcut.
construction(pointcut) all join points where an instance of the given class(es) is constructed.
destruction(pointcut) all join points where an instance of the given class(es) is destructed.
get(pointcut) provides all join points where a global variable or data member in the pointcut is read.
set(pointcut) provides all join points where a global variable or data member in the pointcut is written.
ref(pointcut) provides all join points where a reference (reference type or pointer) to a global variable or data member in the pointcut is created.

pointcut may contain function, variable, namespace or class names. A namespace or class name is equivalent to the names of all functions and variables defined within its scope combined with the | operator (see below).

Control Flow
cflow(pointcut) captures join points occurring in the dynamic execution context of join points in the pointcut. The argument pointcut is forbidden to contain context variables or join points with runtime conditions (currently cflow, that, or target).

Types
base(pointcut) returns all base classes resp. redefined functions of classes in the pointcut
derived(pointcut) returns all classes in the pointcut and all classes derived from them resp. all redefined functions of derived classes

Scope
within(pointcut) filters all join points that are within the functions or classes in the pointcut
member(pointcut) maps the scopes given in pointcut to any contained named entities. Thus a class name for example is mapped to all contained member functions, variables and nested types.
The JoinPoint-API is provided within every advice code body by the built-in `JoinPoint-API` for Advice Code.

Named Pointcuts and Attributes

Pointcut expressions can also refer to user-defined pointcuts.

```cpp
Pointcut @& Pointcut (N(N) -> N, (CC) -> C)
```

Named Pointcuts and Attributes

Named pointcuts and attributes are used to pass the name of a context variable to which the context information is bound. In this case the type of the variable is used for the type matching.

Runtime Functions and State

Runtime Type Information

JoinPoint-API for Slices

The JoinPoint-API is provided within introduced slices by the built-in class `JoinPoint` (state of target class before introduction).

Example (simple tracing aspect)

```cpp
aspect Tracing {
  advice execution("% Business::%(...)") : before() {
    cout << "before " << __FILE__ << __LINE__ << endl;
  }
};
```


(c) Copyright 2021, AspectC++ developers. All rights reserved.

† support for template instance matching is an experimental feature

‡ This feature has limitations. Please see the AspectC++ Language Reference.

‡‡ http://www.aspectc.org

†† https://mentorembedded.github.io/cxx-abi/abi.html#mangling

†‡ C, C++, C, C++, C++, C++, C, C++, C: Code (any, only Call, only Builtin, only Execution, only object Construction, only object Destruction, only get, only set, Only ref) N, N, N, N, N, N: Names (any, only Namespace, only Class, only Function, only Variables, only Type)