```
with BASE,
     ENTRIES,
     CONVENTION,
     EXCEPTIONS,
     CONTROL,
     PROGRAM,
     ALLOCATOR,
     OPS,
     CHILD,
     FRAME,
     MODULE,
     RENDEZVOUS,
     INTERPROCESSOR,
     INSTRUCTION_UNIT;

separate (EXECUTION_UNIT)
package body SUBPROGRAM_OP is

    use CODE, STACKS; -- required for + and -;


    procedure DECL (START      : CODE.SEGMENT_INDEX;
                    VISIBILITY : CODE.VISIBILITY) is
        CURRENT_SEGMENT    : CODE.SEGMENT_NAME;
        FIRST_INSTRUCTION  : CODE.QUAD_WORD;
        ACTIVE_VISIBILITY  : CODE.VISIBILITY;
        PROC_VARIABLE      : STACKS.CONTROL_QUAD
                                 (OF_KIND => STACKS.PROCEDURE_VAR);
    begin
        CURRENT_SEGMENT   := INSTRUCTION_UNIT.CURRENT_PROGRAM_COUNTER().SEGMENT;

        FIRST_INSTRUCTION := PROGRAM.MEMORY.READ
                                 (PROGRAM.MEMORY.ADDRESS'
                                         (SEGMENT =>CURRENT_SEGMENT,
                                          OFFSET  =>START));
        if VISIBILITY = CODE.IS_VISIBLE then
            ACTIVE_VISIBILITY := CODE.IS_INACTIVE;
        else
            ACTIVE_VISIBILITY := VISIBILITY;
        end if;

        PROC_VARIABLE := STACKS.CONTROL_QUAD'
                        (OF_KIND      => STACKS.PROCEDURE_VAR,
                         PROC_ENVIRON => STACKS.CONTROL_REFERENCE'
                                         (FRAME.CURRENT_STACK (),
                                          FRAME.INNER_FRAME ()),
                         PROC_VISIBLE => ACTIVE_VISIBILITY,
                         PROC_SCOPE   => FRAME.OUTER_FRAME (),
                         PROC_LEX     => CODE.LEXICAL_LEVEL(INTEGER
                                         (FRAME.LEX_LEVEL ()) + 1),
                         PROC_FIRST   => FIRST_INSTRUCTION (4),
                         PROC_SEGMENT => CURRENT_SEGMENT,
                         PROC_ADDRESS => START);
        CONTROL.PUSH (PROC_VARIABLE);
    end DECL;
```

```
procedure CALL (SUBPROGRAM_VALUE : STACKS.CONTROL_QUAD
                                    (OF_KIND => STACKS.PROCEDURE_VAR)) is
begin
    if SUBPROGRAM_VALUE.PROC_FIRST.FOR_OP /= CODE.OP then
        FRAME.PUSH (SUBPROGRAM_VALUE);
        INSTRUCTION_UNIT.SET_PROGRAM_COUNTER
                        (CODE.ADDRESS'
                            (SEGMENT   => SUBPROGRAM_VALUE.PROC_SEGMENT,
                             QUAD_WORD => SUBPROGRAM_VALUE.PROC_ADDRESS,
                             HALF_WORD => 5));
    end if;
    INSTRUCTION_UNIT.INSERT (SUBPROGRAM_VALUE.PROC_FIRST);
end CALL;


procedure CALL (SUBPROGRAM_REF: CODE.OBJECT_REFERENCE) is
    SUBPROGRAM_VAR : STACKS.CONTROL_QUAD;
begin
    SUBPROGRAM_VAR := FRAME.READ (SUBPROGRAM_REF);
    case SUBPROGRAM_VAR.OF_KIND is
        when STACKS.PROCEDURE_VAR =>
            CALL (SUBPROGRAM_VAR);

        when STACKS.ACCEPT_VAR =>
            RENDEZVOUS.DO_ACCEPT (SUBPROGRAM_VAR);

        when STACKS.SELECT_VAR =>
            RENDEZVOUS.DO_SELECT (SUBPROGRAM_VAR);

        when others =>
            ERROR.ILLEGAL_INSTRUCTION;
    end case;
end CALL;


procedure PUSH_WORD (VALUE : INTEGER) is
begin
    CONTROL.PUSH (OPS.MAKE_DISCRETE (STACKS.INTEGER_VAR, VALUE));
end PUSH_WORD;
```

```
procedure EXIT_BLOCK_DEALLOCATE (REPEAT_INSTR : CODE.INSTRUCTION;
                                 PARAMETER    : in out INTEGER;
                                 COMPLETE     : out BOOLEAN) is
    NEXT_CHILD  : STACKS.CHILD.WORD;
    CHILD_FOUND : BOOLEAN;
    PARAM_WORD  : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
begin
    if CHILD.DEPENDENTS_DONE () then
        if MODULE.BLOCKED_STATE () /=
                              STACKS.MODULE.WAITING_FOR_CHILDREN then
            PUSH_WORD (PARAMETER);
        end if;
        CHILD.NEXT_TO_DEALLOCATE (NEXT_CHILD, CHILD_FOUND);
        if CHILD_FOUND then
            case NEXT_CHILD.OF_KIND is
                when STACKS.CHILD.MODULE =>
                    INSTRUCTION_UNIT.INSERT (REPEAT_INSTR);
                    INTERPROCESSOR.SEND (INTERPROCESSOR.MESSAGE'
                            (OF_KIND   => INTERPROCESSOR.TERMINATE_MODULE,
                             ADDRESSEE => NEXT_CHILD.MODULE_NAME,
                             ERROR     => CONVENTION.NO_ERROR));

                when STACKS.CHILD.COLLECTION =>
                    ALLOCATOR.FREE (NEXT_CHILD.HEAP_NAME);

                when STACKS.CHILD.IMPORT =>
                    ALLOCATOR.FREE_IMPORT_SPACE
                                  (FOR_MODULE  => FRAME.CURRENT_STACK (),
                                   TYPE_OFFSET => NEXT_CHILD.DECL_SITE);

                when others =>
                    raise EXCEPTIONS.IN_SIMULATION.SIMULATION_ERROR;
            end case;
            COMPLETE  := FALSE;
        else
            PARAMETER := INTEGER (CONTROL.POP().DATA_ITEM.VAL.INT_VAL);
            COMPLETE  := TRUE;
        end if;
    else
        INSTRUCTION_UNIT.INSERT (REPEAT_INSTR);
        PUSH_WORD (PARAMETER);
        MODULE.SAVE (BLOCK_STATE => STACKS.MODULE.WAITING_FOR_CHILDREN);
        COMPLETE := FALSE;
    end if;
end EXIT_BLOCK_DEALLOCATE;
```

```
procedure POP_LEVEL (CURRENT_COUNT : in out INTEGER;
                     REPEAT_TYPE   : CODE.OP_CODE) is
    REPEAT_INSTRUCTION : CODE.INSTRUCTION (FOR_OP => REPEAT_TYPE);
begin
    FRAME.POP (PARAMETER_COUNT  => 0,
               LOAD_INSTRUCTION => FALSE);
    CURRENT_COUNT := CURRENT_COUNT - 1;
    if CURRENT_COUNT /= 0 then
        REPEAT_INSTRUCTION.LEVEL_COUNT :=
                    CODE.LEXICAL_LEVEL (CURRENT_COUNT);
        INSTRUCTION_UNIT.INSERT (REPEAT_INSTRUCTION);
    end if;
end POP_LEVEL;


procedure EXIT_PROCEDURE (PARAMS : CODE.IN_PARAMETER_COUNT) is
    NUMBER_PARAMS      : INTEGER := INTEGER (PARAMS);
    DEALLOCATION_DONE : BOOLEAN;
begin
    if not FRAME.HAS_CHILDREN () or else FRAME.EXECUTING_INIT () then
        FRAME.POP (PARAMS);
        OPS.COMPLETE_EXPRESSION;
    else
        EXIT_BLOCK_DEALLOCATE (REPEAT_INSTR => CODE.INSTRUCTION'
                                    (FOR_OP       => CODE.EXIT_PROC,
                                     PARAM_COUNT => 0),
                               PARAMETER     => NUMBER_PARAMS,
                               COMPLETE      => DEALLOCATION_DONE);
        if DEALLOCATION_DONE then
            FRAME.POP (CODE.IN_PARAMETER_COUNT (NUMBER_PARAMS));
            OPS.COMPLETE_EXPRESSION;
        end if;
    end if;
end EXIT_PROCEDURE;

procedure EXIT_FUNCTION (PARAMS : CODE.IN_PARAMETER_COUNT) is
    RESULT             : STACKS.CONTROL_QUAD;
    NUMBER_PARAMS      : INTEGER := INTEGER (PARAMS);
    DEALLOCATION_DONE : BOOLEAN;
begin
    if not FRAME.HAS_CHILDREN () or else FRAME.EXECUTING_INIT () then
        RESULT := CONTROL.POP ();
        FRAME.POP (PARAMS);
        CONTROL.PUSH (RESULT);
    else
        EXIT_BLOCK_DEALLOCATE (REPEAT_INSTR => CODE.INSTRUCTION'
                                    (FOR_OP       => CODE.EXIT_FUNC,
                                     PARAM_COUNT => 0),
                               PARAMETER     => NUMBER_PARAMS,
                               COMPLETE      => DEALLOCATION_DONE);
        if DEALLOCATION_DONE then
            RESULT := CONTROL.POP ();
            FRAME.POP (CODE.IN_PARAMETER_COUNT (NUMBER_PARAMS));
            CONTROL.PUSH (RESULT);
        end if;
    end if;
end EXIT_FUNCTION;
```

```
procedure POP_PROC  (LEVELS : CODE.LEXICAL_LEVEL) is
    NUMBER_LEVELS      : INTEGER := INTEGER (LEVELS);
    DEALLOCATION_DONE : BOOLEAN;
begin
    if not FRAME.HAS_CHILDREN () then
        POP_LEVEL (CURRENT_COUNT => NUMBER_LEVELS,
                   REPEAT_TYPE   => CODE.POP_PROC);
        OPS.COMPLETE_EXPRESSION;
    else
        EXIT_BLOCK_DEALLOCATE (REPEAT_INSTR => CODE.INSTRUCTION'
                                        (FOR_OP       => CODE.POP_PROC,
                                         LEVEL_COUNT => CODE.LEXICAL_LEVEL
                                                          (NUMBER_LEVELS)),
                               PARAMETER     => NUMBER_LEVELS,
                               COMPLETE      => DEALLOCATION_DONE);
        if DEALLOCATION_DONE then
            POP_LEVEL (CURRENT_COUNT => NUMBER_LEVELS,
                       REPEAT_TYPE   => CODE.POP_PROC);
            OPS.COMPLETE_EXPRESSION;
        end if;
    end if;
end POP_PROC;


procedure POP_FUNC (LEVELS : CODE.LEXICAL_LEVEL) is
    RESULT             : STACKS.CONTROL_QUAD;
    NUMBER_LEVELS      : INTEGER := INTEGER (LEVELS);
    DEALLOCATION_DONE : BOOLEAN;
begin
    if not FRAME.HAS_CHILDREN () then
        RESULT := CONTROL.POP ();
        POP_LEVEL (CURRENT_COUNT => NUMBER_LEVELS,
                   REPEAT_TYPE   => CODE.POP_FUNC);
        CONTROL.PUSH (RESULT);
    else
        EXIT_BLOCK_DEALLOCATE (REPEAT_INSTR => CODE.INSTRUCTION'
                                        (FOR_OP       => CODE.POP_FUNC,
                                         LEVEL_COUNT => CODE.LEXICAL_LEVEL
                                                          (NUMBER_LEVELS)),
                               PARAMETER     => NUMBER_LEVELS,
                               COMPLETE      => DEALLOCATION_DONE);
        if DEALLOCATION_DONE then
            RESULT := CONTROL.POP ();
            POP_LEVEL (CURRENT_COUNT => NUMBER_LEVELS,
                       REPEAT_TYPE   => CODE.POP_FUNC);
            CONTROL.PUSH (RESULT);
        end if;
    end if;
end POP_FUNC;
```

```
function EXTRACT_COUNT (FROM_ENTRY : STACKS.CONTROL_QUAD) return INTEGER is
begin
    case FROM_ENTRY.OF_KIND is
        when STACKS.ENTRY_VAR  =>
            return INTEGER (FROM_ENTRY.PARAMETER_SIZE);

        when STACKS.FAMILY_VAR =>
           return INTEGER (FROM_ENTRY.FAMILY_PARAMS);

        when others =>
            raise EXCEPTIONS.IN_SIMULATION.SIMULATION_ERROR;
    end case;
end EXTRACT_COUNT;


function POP_PARAMS (COUNT : INTEGER) return ENTRIES.PARAM_BLOCK is
    BLOCK : ENTRIES.PARAM_BLOCK (OF_SIZE => COUNT);
begin
    for I in 1 .. COUNT loop
        BLOCK.ELEMENT (I) := CONTROL.POP ();
    end loop;
    return BLOCK;
end POP_PARAMS;
```

```
    procedure EXIT_ACCEPT (PARAMS : CODE.IN_PARAMETER_COUNT) is

        use STACKS;

        ACCEPT_LINK   : STACKS.CONTROL_QUAD
                              (OF_KIND => STACKS.ACCEPT_LINK);
        CALLER        : STACKS.NAME;
        ENTRY_SITE    : CODE.OBJECT_REFERENCE;
        ENTRY_VAR     : STACKS.CONTROL_QUAD;
                           -- (OF_KIND => STACKS.ENTRY_VAR | STACKS.FAMILY_VAR);
        TOTAL_PARAMS  : INTEGER;
        OUT_BLOCK     : ENTRIES.PARAM_BLOCK;
    begin
        FRAME.POP (PARAMETER_COUNT  => 0,
                   LOAD_INSTRUCTION => TRUE);
        ACCEPT_LINK   := CONTROL.POP ();
        CONTROL.POP (DOWN_TO => FRAME.CURRENT_OFFSET () -
                                        STACKS.CONTROL_DISPLACEMENT (PARAMS));
        CALLER        := MODULE.IN_RENDEZVOUS ();
        ENTRY_SITE    := CODE.OBJECT_REFERENCE'
                                        (LEVEL  => 0,
                                         OFFSET => MODULE.ACCEPT_ENTRY ());
        ENTRY_VAR     := FRAME.READ (ENTRY_SITE);
        TOTAL_PARAMS  := EXTRACT_COUNT (ENTRY_VAR);
        OUT_BLOCK     := POP_PARAMS (COUNT => TOTAL_PARAMS - INTEGER (PARAMS));
        MODULE.END_RENDEZVOUS (FROM_LINK => ACCEPT_LINK);
        INTERPROCESSOR.SEND (INTERPROCESSOR.MESSAGE'
                                  (OF_KIND     => INTERPROCESSOR.END_RENDEZVOUS,
                                   ADDRESSEE   => CALLER,
                                   ERROR       => CONVENTION.NO_ERROR,
                                   OUT_PARAMS  => OUT_BLOCK));
    end EXIT_ACCEPT;

end SUBPROGRAM_OP;
```