**TYPE EQUIVALENCE**

1) Coercion
2) Casting
3) Conversion

**COERCION**

- Operators require their operands to be of a certain type (similarly expressions require their arguments to be of the same type).
- In some cases it may be appropriate, when a compiler finds an operand that is not quite of the “right” type to convert the operand so that it is of the right type.
- This is called coercion.
- C supports coercion, Ada does not.

**CASTS**

- It is sometimes necessary for a programmer to force a coercion.
- This is referred to as a cast.
- This is supported in C (but not Ada).
- Example:

```c
void main(void) {
  float j = 2.0;
  int i;
  i = (int) (j * 0.5);
}
```

**CONVERSION**

- Where casting is not supported the most common alternative is explicit conversion. This is supported by Ada.
- Ada conversions have the following form:

```
T(N)
```

where T is the name of a numeric type and N has another numeric type. The result is of type T.
- The distinction between conversion and casting is that the latter relies on the compiler’s coercion mechanism to achieve the end result while a conversion obtains the desired result explicitly through a call to a conversion function.

**COMP205 IMPERATIVE LANGUAGES**

10. PROGRAM CONSTRUCTS 1
    (SELECTION)

1. Program Statements
2. Assignment
3. Sequences
4. Selection
   a) If statements
   b) Case statements

**STATMENTS**

- Statements are the commands in a language that perform actions and change the *state*.
- The *state* of a program is described by the values held by its variables at a specific point during its execution.
- Example statements:
  1) Assignment *statements*, which change the values of variables.
  2) Sequence *statements*, which define the flow of a program.
  3) Selection *statements*, which define alternative courses of action.
  4) Repetition *statements*, which define program loops.
- The last three are often referred to as program constructs.
- Statements are usually separated by some operator (symbol), in Ada and C this a ;.
ASSIGNMENT STATEMENTS

- The value of a variable can always be modified using an assignment statement:
  \[ i := 2; \]

- The source of a value in an assignment is usually an expression referred to as an assignment expression.

- Some imperative languages support multiple assignment:
  \[ i, j := 2; \]

THE C POST-INCREMENT AND PRE-INCREMENT OPERATORS

- Certain assignment statements such as:
  \[ j := j+1; \]
  \[ k := k-1; \]
  \[ J := J+1; \]
  \[ K := K-1; \]

- are so common that some imperative languages (but not Ada) provide a short hand notation for them:
  \[ i++; \]
  \[ i--; \]

- The operators are referred to as the post-increment and post-decrement operators.

PROGRAM CONSTRUCTS

- All programming language utilise program constructs.
- Program constructs are statements like any other.
- In imperative languages they are used to control the order (flow) in which statements are executed (or not executed).
- There are a number of recognised basic programming constructs that can be classified as follows:
  1) Sequences
  2) Selection
  3) Repetition
- To which we can also add routine invocation.

SEQUENCES

- A sequence statement tells the processor which statement is to be executed next.
- By default, in imperative languages, this is the statement following the current statement (or the first statement in the program).
- If we wish to "jump" to some other statement we can use a goto statement.
- This was popular in the early days of computer programming (1950s to early 1960's).
- BASIC goto statement: \[ \text{GOTO 400} \]
- Also encouraged in languages such as Fortran and Cobol.

GOTO STATEMENT CONSIDERED HARMFULL (Dijkstra 1968)

- goto statements make it difficult to determine the state of variables at a given point during processing. As a result errors are often obscure and difficult to locate.
- For this reason some imperative languages, for example Modula-2, have abandoned the goto statement.
- However there are still some legitimate reasons why a goto may be desirable, for example to facilitate error handling or to terminate a deeply nested sequence of loops.
- Both Ada and C support a goto statement but it is best avoided.

ROUTINE INVOCATION

- Routine (or procedure) invocation is particular to imperative languages.
- When a sequence of statements forms a conceptual unit about which it is possible and useful to think and reason in isolation it is convenient to encapsulate the sequence in a named routine (procedure) and to replace it with a procedure call in the original code.
- Unlike a goto statement, routine invocation guarantees that the flow of control will eventually return to the point from which the routine was called (the procedure call).
SELECTION

• A selection statement provides for selection between alternatives.

• We can identify two basic categories of selection construct:
  1) If statements
  2) Case statements

• To which we can add *pattern matching* although imperative languages do not usually support this (logic and some modern functional languages do).

IF-ELSE SELECTION STATEMENTS

• An *if* statement, sometimes referred to as a conditional, can be used in two forms:
  1) If *X* then *Y*
  2) If *X* then *Y* else *Z*

• Where *X* is a *Boolean expression* (or a sequence of Boolean expressions) and *Y* and *Z* are program statements of some description.

• We can also identify a number of variations of the above which are particular to individual programming languages. For example:

  ```
  if ... then ...elseif ...
  then ... elseif ...
  then ... else ...
  ```

IF-ELSE STATEMENTS

IF-ELSE EXAMPLES

C

```c
if (x < 0) {
  y = -x;
  z = 10+x;
}
else {
  y = x;
  z = 10-x
}
```

Ada

```ada
if X < 0 then
  Y := -X;
  Z := 10+X;
else
  Y := X;
  Z := 10-X;
end if;
```

Pascal

```pascal
if x < 0 then begin
  y := -x;
  z := 10+x;
end;
else begin
  y := x;
  z := 10-x
end;
```

Modula-2

```modula-2
IF (x < 0) THEN
  y := -x;
  z := 10+x
ELSE
  y := x;
  z := 10-x
END
```

IF-ELSE SUMMARY

```
Condition
(Boolean expression)
--------------------
T

Action 1
(Statements)
```

```
F

Action 2
(Statements)
```
NESTED “IF-ELSE” C EXAMPLE

```c
#include<stdio.h>
void main(void) {
  int number;
  char type[8];
  scanf("%d",&number);
  if (number < 100) {
    if (number < 10) strcpy(type,"small");
    else strcpy(type,"medium");
  } else strcpy(type,"large");
  printf("%d is %s\n",number,type);
}
```

CASE SELECTION STATEMENTS

- Generally speaking an "if ... then ... else ..." statement supports selection from only two alternatives; we can of course nest such statements, but it is usually more succinct to use a case statement.
- Case statements allow selection from many alternatives.
- Selections may be made according to:
  a) A distinct value of a given selector,
  b) An expression involving the selector, or
  c) A default value.

CASE SELECTION STATEMENTS Cont.

- A selector must be of a discrete type (typically an integer, a character or an enumerated type).
- C only supports selection by distinct value or default (uses the keyword default).
- Ada, in addition to selection by distinct value and default (keyword others) supports selection according to:
  a) A number of Alternatives (each separated by a bar |), and
  b) Ranges expressed using the .. operator.
- Neither Ada or C support selection through expressions involving the selector.

CASE STATEMENTS

```c
# include <stdio.h>

procedure CASE_EXAMPLE is SELECTOR : NATURAL;
begin
  PUT_LINE("Input selector");
  GET(SELECTOR);
  case SELECTOR is
    when 0 =>
      PUT_LINE("0 value");
    when 1..3 =>
      PUT_LINE("range 1..3");
    when 4 | 5 =>
      PUT_LINE("4 or 5");
    when others =>
      PUT_LINE("other");
  end case;
end CASE_EXAMPLE;
```

ADA CASE EXAMPLE

1) Single value
2) Range of values
3) Alternatives
4) Default
```c
typedef enum {MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
               FRIDAY, SATURDAY, SUNDAY} WEEK_T;

void main(void) {
    WEEK_T day;
    printf("Input day of the week, 0 = Monday, Etc.\n");
    scanf("%d", &day);
    switch (day) {
        case 0:
        case 1:
        case 2:
        case 3:
        case 4:
            printf("Week day\n");
            break;
        case 5:
            printf("Saturday\n");
            break;
        case 6:
            printf("Sunday\n");
            break;
        default:
            printf("Error\n");
    }
}
```

**C CASE EXAMPLE**

1) Range of values
2) Single values
3) Default

**SUMMARY**

1. Sequences
2. Selection
   a) If statements
   b) Case statements