

---

AdaCore

---

## Project Properties Lab

- Create new project file in an empty directory
- Specify source and output directories
  - Use source files from the 030\_project\_properties directory (under source)
  - Specify where object files and executable should be located
- Build and run executable (pass command line argument of 200)
  - Note location of object files and executable
  - Execution should get Constraint\_Error

## Directories Solution

- Project File

```
project Lab is
  for Source_Dirs use ("source/030_project_properties");
  for Main use ( "main.adb" );
  for Object_Dir use "obj";
  for Exec_Dir use "exec";
end Lab;
```

- Executable Output

...

```
41      267914296
42      433494437
43      701408733
44      1134903170
45      1836311903
```

```
raised CONSTRAINT_ERROR : fibonacci.adb:16 overflow check failed
```

## Project Properties Lab - Switches

- Modify project file to disable overflow checking
  - Add the Compiler package

- 
- Insert `Default_Switches` attribute for Ada in `Compiler` package
  - Set switch `-gnato0` in the attribute
    - \* Disable overflow checking
  - Build and run again
    - Need to use switch `-f` on command line to force rebuild
      - \* (Changes to GPR file do not automatically force recompile)
    - No `Constraint_Error`
      - \* But data doesn't look right due to overflow issues

## Switches Solution

- Project File

```
project Lab is
  for Source_Dirs use ("source/030_project_properties");
  for Main use ( "main.adb" );

  package Compiler is
    for Default_Switches ("Ada") use ("-gnato0");
  end Compiler;
  ...
end Lab;
```

- Executable Output

```
...
43      701408733
44     1134903170
45     1836311903
46    -1323752223
47      512559680
48     -811192543
49     -298632863
50    -1109825406
...
```

---

## Project Properties Lab - Naming

- Modify project file to use naming conventions from a different compiler
  - Change source directories to point to naming folder
  - File naming conventions:
    - \* Spec: <unitname>[.child].1.adb
    - \* Body: <unitname>[.child].2.adb
  - Remember to fix executable name
- Build and run again
  - *Note: Accumulator uses more bits, so failure condition happens later*

## Naming Solution

- Project File

```
project Lab is
  for Source_Dirs use ("source/030_project_properties/naming");

  package Naming is
    for Casing use "lowercase";
    for Dot_Replacement use ".";
    for Spec_Suffix ("Ada") use ".1.adb";
    for Body_Suffix ("Ada") use ".2.adb";
  end Naming;

  for Main use ( "main.2.adb" );
  ...
end Lab;
```

- Executable Output

```
...
88  1779979416004714189
89  2880067194370816120
90  4660046610375530309
91  7540113804746346429
92  -6246583658587674878
93  1293530146158671551
```

---

```
94  -4953053512429003327
95  -3659523366270331776
96  -8612576878699335103
...
```

## Project Properties Lab - Conditional

- Modify project file to select precision via compiler switch
  - `conditional` folder has two more package bodies using different accumulators
  - Read a variable from the command line to determine which body to use
    - \* Hint: Naming will need to use a case statement to select appropriate body
- Build and run again
  - Hint: Name used in **external** call must be same casing as in `gprbuild` command, i.e.
    - \* `external ("FooBar");` means `gprbuild -XFooBar...`

## Conditional Solution

- Project File

```
project Lab is

  for Source_Dirs use ("source/030_project_properties/naming",
                      ↪ "source/030_project_properties/conditional");

  type Precision_T is ( "unsigned", "float", "default" );
  Precision : Precision_T := external ( "PRECISION", "default");

  package Naming is
  ...
    case Precision is
    when "unsigned" =>
      for Body ("Fibonacci") use "fibonacci.unsigned";
    when "float" =>
      for Body ("Fibonacci") use "fibonacci.float";
    when "default" =>
      for Body ("Fibonacci") use "fibonacci.2.ad";
```

---

```
    end case;  
end Naming;
```

```
    ...  
end Lab;
```

- Executable Output

```
1    1.000000000000000E+00  
2    2.000000000000000E+00  
3    3.000000000000000E+00  
4    5.000000000000000E+00  
5    8.000000000000000E+00  
6    1.300000000000000E+01  
7    2.100000000000000E+01  
8    3.400000000000000E+01  
9    5.500000000000000E+01  
10   8.900000000000000E+01  
... 
```