Advanced practical **Programming** for Scientists

Thorsten Koch

Zuse Institute Berlin

TU Berlin

WS2014/15

Choices

`int verb_level` vs `enum Verbosity`

problem with enum is that actually we are hardly ever comparing == but usually <= >=.
Where is the extra benefit for handling an `enum` compared to the `int`?

where to put the definition: bip.h or ex7.h

where to store it:

- global variable (threads? problem?)
- pass through all functions
- store in central data structure like BIP

Belongs in a sense to ex7.h but then we would have a cycle between bip.h and ex7.h

Extra module with static variable, called from everybody.

Could have verbosity.h with just the defines.

Choices

Only have ac no ar. Since we are changing and reordering the input file we want to have the original (or not)?

Again, do these belong into bip.h ?

```
double    min_coef_val;
double    max_coef_val;
int       verb_level;
int       read_rows;
```

```
      if (x & updatemask)
      {
         for(k = 0; k < rows; k++)
            r[k] += modcol[k];
      }
      else /* bit changed from 1 to 0 */
      {
         for(k = 0; k < rows; k++)
            r[k] -= modcol[k];
      }
```

Vs.

```
      for(k = 0; k < rows; k++)
         if (x & updatemask)
            r[k] += modcol[k];
         else
            r[k] -= modcol[k];
```

Vs.

```
      for(k = 0; k < rows; k++)
         r[k] += (x & updatemask) ? modcol[k] : -modcol[k];
```

Changes

- Introduced `ac`.
  Filled in preprocess() after reading, instead of changing and copying the data in ex7 main().
- Put the enumerate() feasibility check into subroutine. Notice: inline.
- enumerate part of BIP. Introduces constructer/deconstructor

Details

bip.h

- Naming: If the function does something there is a verb.
- If it just reports a property of the data structure the implicit 'get' is omitted.

Problem BIP has three stages:

- Allocated
- ar read in
- preprocessing done and ac build

C99:

```
double*        r = malloc((size_t)rows * sizeof(*r));
double         r[rows];
```

Coverage test

Use –DNDEBUG yes or no?

No: Easier to get 100%. Can be difficult to produce coverage for certain assert related checking function as the situation should/could never happen.

Yes. Checking code has to be part of coverage test. Otherwise it is unclear whether the checking actually happens. Also it may be unclear whether the checks are correct.

Problem: dependencies

Code gets hard to change because of many dependencies.

- splitline should be named split_string.
- Change of the signature and sematics of a function including errors. -> Lots of changes in code, documentation, tests.
- While code will mostly automatically stay coherend (compiler will find not adapted signatures) **documentation** and tests might not.

=> Once it is finished and polished you do not want to change it anymore. Support through DevEnv is needed that take care automatically to allow refactoring.

Defensive Programming

```
408          :
409          :      default :
410        0 :          abort();
411          :      }
412          :  }
```

Good defensive programming.

Compiler might or might not detect not reachable. Coverage will detect.
Defense is against changes in the code that make it reachable again.

Short circuit evaluation

```
int k;

for(k = 0; (k < bip->equs) && (r[k] == 0.0); k++)
   ;

if (k == bip->equs)
   for(; (k < bip->rows) && (r[k] <= 0.0); k++)
      ;

if (k < bip->rows)
   return 0;

(*report_sol)(bip, x);
solution_count++;
```

```
k := 1;
if (k < Equs) then
   loop
      if r(k) /= 0.0 then feasible := false; exit; end if;
      if k = Equs then exit; end if;
      k := k + 1;
   end loop;
end if;

if feasible then
   if k < Rows then
      loop
         if r(k) > 0.0 then feasible := false; exit; end if;
         if k = Rows then exit; end if;
         k := k + 1;
      end loop;
   end if;
   if feasible then
      Report_solution(Inst, x);
      Solution_Count := Solution_Count + 1;
   end if;
end if;
```

Ariane 5

On 4 June 1996 the maiden flight of the Ariane 5 launcher ended in a failure, about 40 seconds after initiation of the flight sequence. At an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded. The failure was caused by "*complete loss of guidance and attitude information*" 30 seconds after liftoff.

The problem was caused by an `Operand Error' in converting data in a subroutine from 64-bit floating point to 16-bit signed integer. One value was too large to be converted, creating the Operand Error. This was not explicitly handled in the program (although other potential Operand Errors were) and so the computer, the Inertial Reference System (SRI) halted, as specified in other requirements. There are two SRIs, one `active', one `hot back-up' and the active one halted just after the backup, from the same problem. Since no inertial guidance was now available, and the control system depends on it, we can say that the destructive consequence was the result of `Garbage in, garbage out' (GIGO). The conversion error occurred in a routine which had been reused from the Ariane 4 vehicle, whose launch trajectory was different from that of the Ariane 5. The variable containing the calculation of Horizontal Bias (BH), a quantity related to the horizontal velocity, thus went out of `planned' bounds (`planned' for the Ariane 4) and caused the Operand Error.

Requirements and Testing

a) The operand range in the module was deliberately not protected;

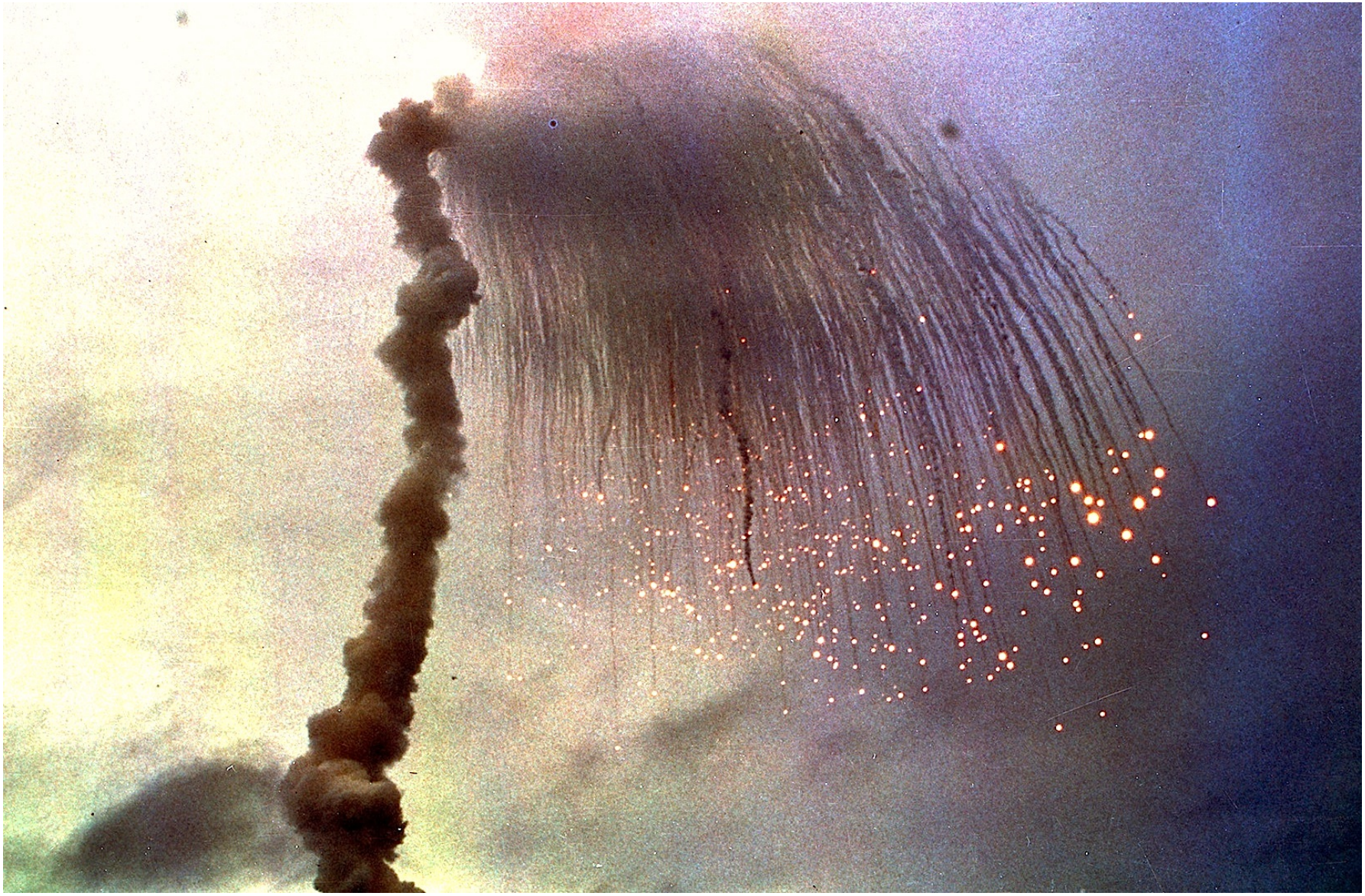
b) this was because engineering analysis for its use in Ariane 4 had shown the operand would never go out of bounds;

c) the range requirement stemming from this analysis was not transferred to the requirements for the Ariane 5;

d) testing was done against requirements

this is more properly classified as a requirements error rather than a programming error. The program was written against Ariane 4 requirements; these requirements were not transferred to the Ariane 5 requirements spec; the Ariane 5 requirements therefore did not state the range requirement; the (implicit in Ariane 5) range requirement was in conflict with the behavior of Ariane 5 (as in fact explicated in other Ariane 5 requirements); requirements came up against behavior and the rocket was destroyed. (It is not surprising that it was a requirements error - over 90% of safety-critical systems failures are requirements errors, according to a JPL study that has become folklore)

Ariane 5



Was has this to do with us?

Your proof / paper will look the same for the same reasons.