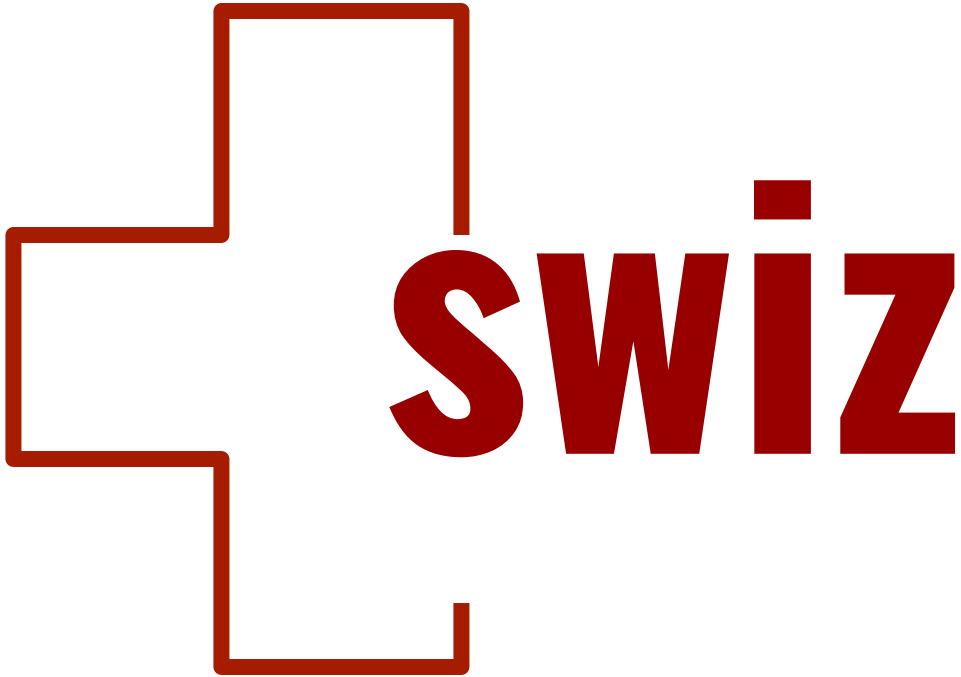


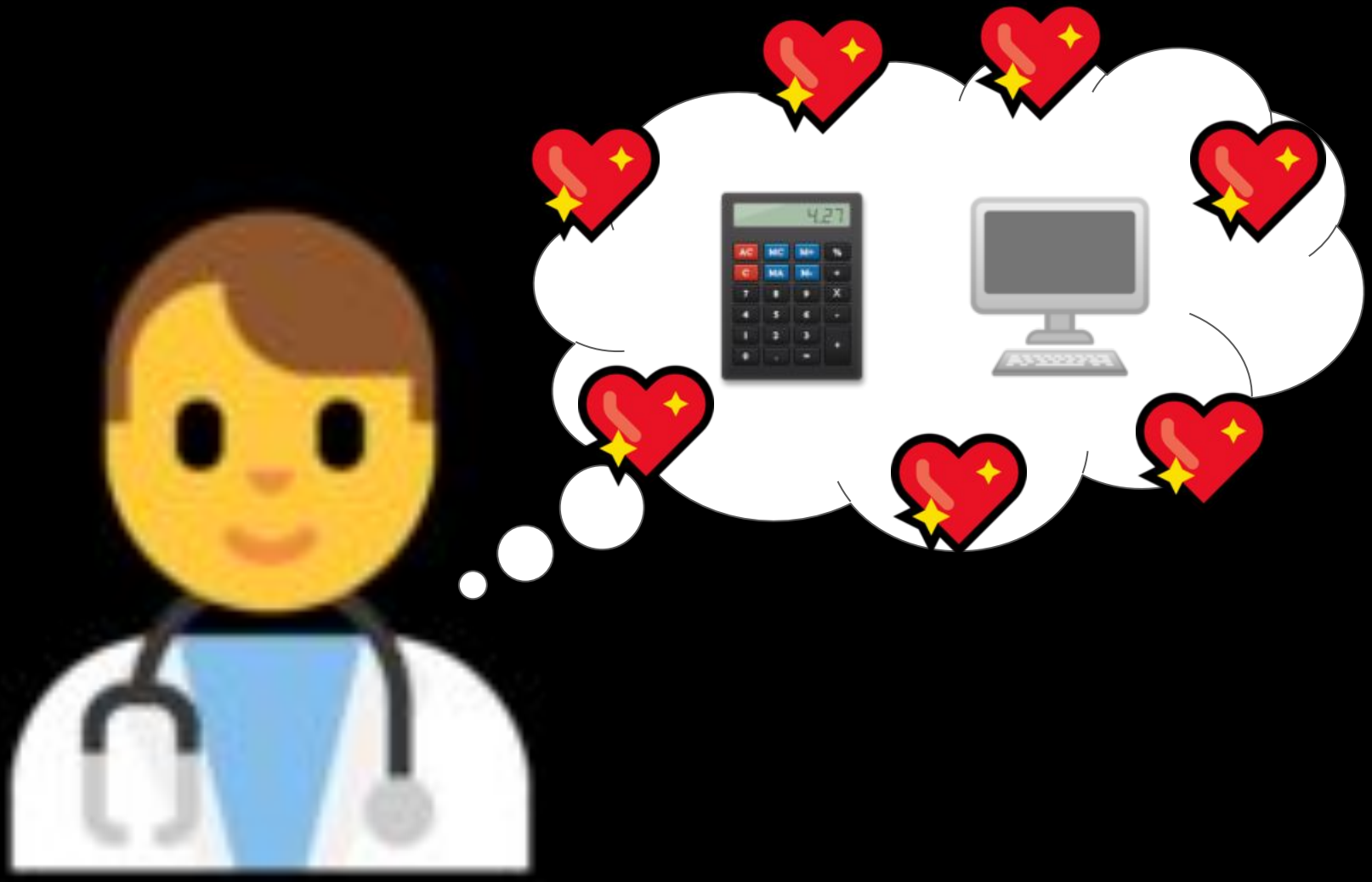
red
test

green
code

blue
refactor

github.com/d-muc/swiz

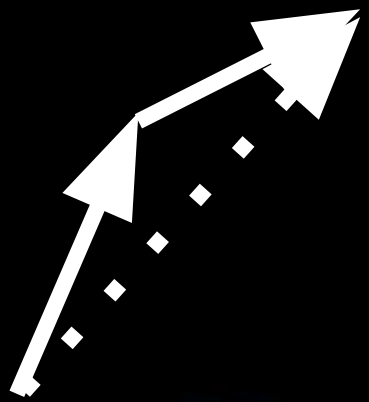
Stefan Rohe, 29.03.2018


4.27
AC
MC
M+
%
C
MR
M-
÷
7
8
9
X
4
5
6
-
1
2
3
+
0
.
=

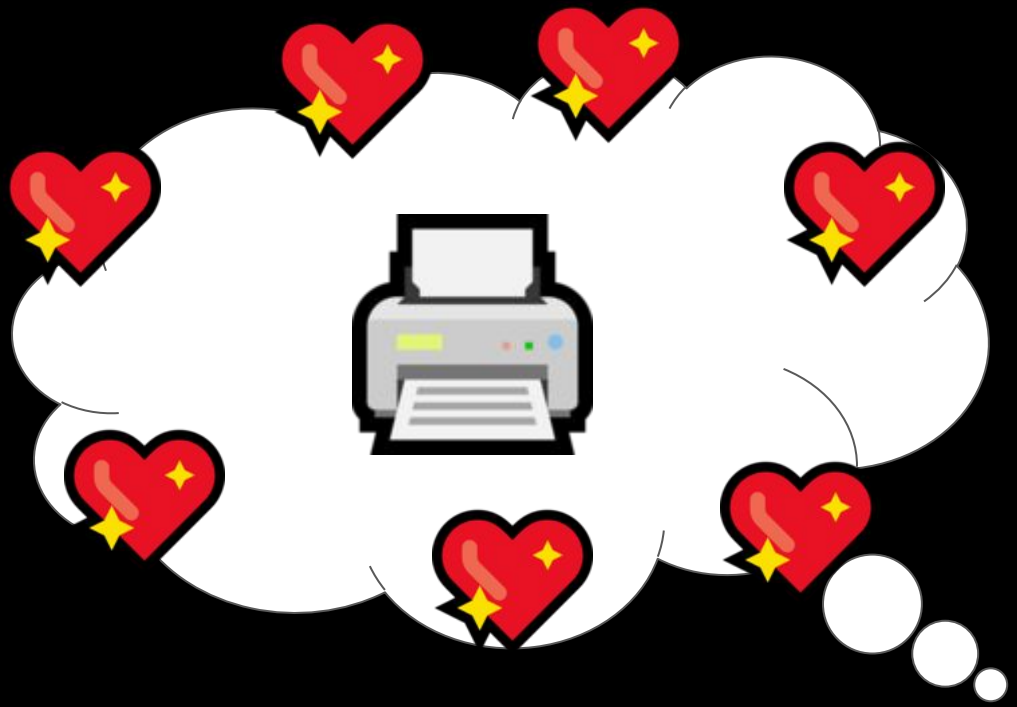
```
#include <catch.hpp>

TEST_CASE("x, y", "[basics]") {
  // arrange
  auto vec = Vector(1.0f, 2.0f);

  // act & assert
  REQUIRE(vec.x == 1.0f);
  REQUIRE(vec.y == 2.0f);
}
```

```
struct Vector {
  float x;
  float y;

  Vector(float x, float y) : x(x), y(y) {}
}
```

```
template <float/double, int n>

// union to access via x, y, z, ...
```

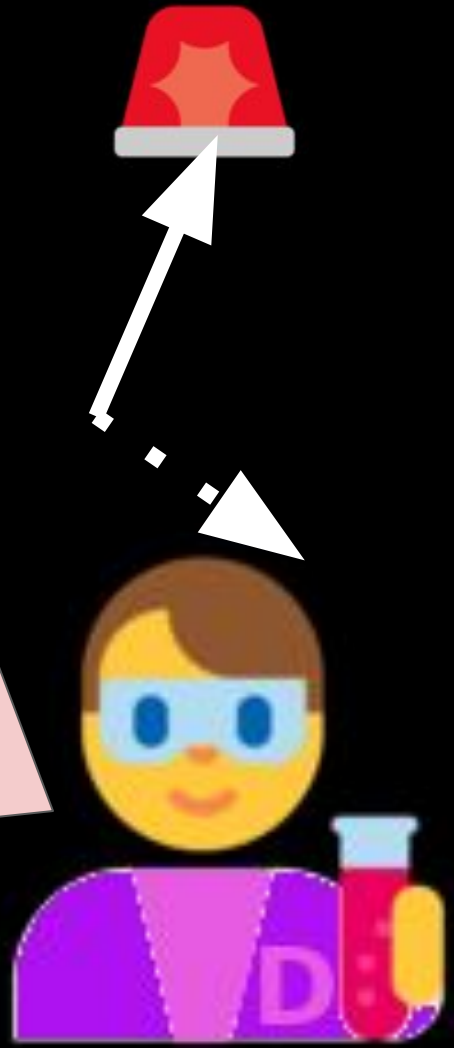


```
TEST_CASE("+", "[basics]") {
  // act
  Vector result =
    Vector(1.0f, 2.0f) + Vector(2.0f, 1.0f);

  // assert
  REQUIRE(result.x == 3.0f);
  REQUIRE(result.y == 3.0f);
}
```



```
unittest {
  // act
  const result = Vector(1, 2) + Vector(2, 1);

  // assert
  assert(result == Vector(3, 3);
}
```

```
unittest {
  // act
  const result = Vector(1, 2).yx;

  // assert
  assert(result == Vector(2, 1);
}
```



swizzling

Wikipedia [https://en.wikipedia.org/wiki/Swizzling_(computer_graphics)]

- Common Operation in GPGPU applications
- A = {1,2,3,4} and components are called x, y, z, w
- B = A.wwxy
- B equals {4, 4, 1, 2}

swizzling in C++

- **Union Trick**
- **Swizzle-Proxy Template**
- **CxxSwizzle (wc -l include/swizzle: 3230 lines)**

```
detail::SwizzleProxy1<Vec1_Base<Data>,Data,0> x, r;
detail::SwizzleProxy1<Vec1_Base<Data>,Data,1> y, g;
detail::SwizzleProxy2<Vec2_Base<Data>,Data,0,0> xx, rr;
detail::SwizzleProxy2<Vec2_Base<Data>,Data,0,1> xy, rg;
detail::SwizzleProxy2<Vec2_Base<Data>,Data,1,0> yx, gr;
detail::SwizzleProxy2<Vec2_Base<Data>,Data,1,1> yy, gg;
detail::SwizzleProxy3<Vec3_Base<Data>,Data,0,0,0> xxx, rrr;
...
```

```
struct Vector(T, size_t Size)
{
  auto swiz(string Str, Args...)(Args args)
    const
  {
    ...
  }
  alias opDispatch = swiz;
}
```

```
auto swiz(string Str, Args...)(Args args)  const {
 enum string prop(size_t i, dchar ch) {
  if (ch == '_')
   return "args[%s]".format(Str[0..i].count('_'));
  if (ch == '0' || ch == '1') return ch.to!string;
   return format!"this.%s"(ch);
 }
 enum props() {
  string[] res;
  foreach (i, ch; Str) {
   res ~= prop(i, ch);
  }
  return res.join(", ");
 }
 return mixin(
  "Vector!(T, %s)(".format(Str.length) ~ props() ~  ")");
}
```

```
unittest {
  // arrange
  const v = Vector(1, 2);

  // act & assert
  assert(v.y0 == Vector(2, 0);
  assert(v.1x == Vector(1, 1);
  assert(v.y_(23) == Vector(2, 23);
}
```

```
unittest {
  // arrange
  auto v1 = Vector(1, 2, 3, 4);
  auto v2 = Vector(4, 3, 2, 1);

  // act & assert
  assert(v1.y0 == Vector(2, 0);
  assert(v1.1x == Vector(1, 1);
  assert(v1.y_(23) == Vector(2, 23);

  assert(v1.wzyx == v2);
  assert(v1.xy == v2.wz);
}
```

github.com/d-muc/swiz