1 Aliasing and the restricted keyword

Aliasing describes the situation of having several means to access the same memory location. This happens often with references and pointers.

Wihout compiler optimization these programs work, but once optimization is enabled, strange effects may happen if the alias is not recognized by the compiler.

For example,

```
void someop(int *a, int *b)
{
    *a = *a + *b;
    *b = *b * 3;

}

int main(int argc, char**)
{
    int x = 1;
    int y = 2;
    someop(x,y);
}
```

For this example the compiler will assume that `*a` and `*b` could be aliases of each other and creates x86 assembler code like,

```
  ; *a = *a + *b
  mov    (%rdx),%eax         ; read *b
  add    %eax,(%rcx)         ; add it to *a
  ; *b = *b * 3
  mov    (%rdx),%eax         ; *read *b again
  lea    (%rax,%rax,2),%eax  ; multiply by 3
  mov    %eax,(%rdx)         ; store to *b
```

This shows that `*b` is assumed to be modified by the write to `*a`. This is not the case in this example, but could happen with,

```
    int x = 3;
    someop(x,x);
```

The `restrict` keyword allows to hint the compiler to drop this assumption for an argument,

```
void someop(int * restrict a, int * restrict b)
{
    *a = *a + *b;
    *b = *b * 3;

}
```

Now the compiler creates different assembler code,

```
  ; *a = *a + *b
  mov    (%rdx),%eax        ; read *b
  add    %eax,(%rcx)        ; add it to *a
  lea    (%rax,%rax,2),%eax ; multiply by 3
  mov    %eax,(%rdx)        ; store to *b
```

This time the second read from `*b` is omitted.

The `restrict` keyword is certainly useful for optimization.

A similar effect can be archived with gcc using the `-fstrict-aliasing` option, which acts like a restrict for every parameter. `-fno-strict-aliasing` assumes the opposite, as if no restrict was ever applied anywhere.