A little bit about optimization,
2d array,
1d array used as 2d,
register
volatile
union

What smart people say about Premature optimization…

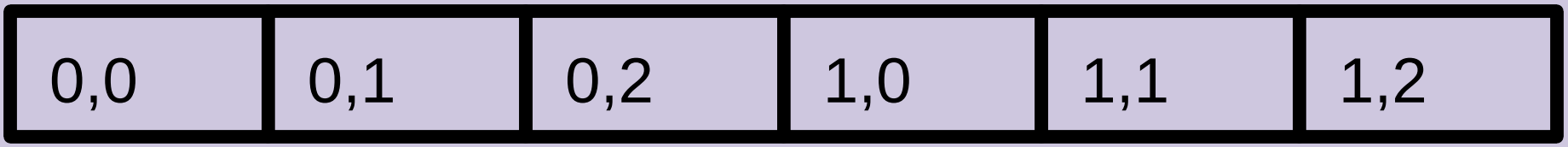
Premature optimization is the root of all evil (or at least most of it) in programming
– Donald Knuth

So, what to do?

- Check if you **need** to optimize
- Remember to **“turn off” debugging** (#define NDEBUG)
- Check what **your compiler** can do for you on **your specific hardware** (-O3 -march=pentium4 -mfpmath=sse, inlining of functions, ...)
- **Profile**: check **where** to optimize
- Use **common techniques** such as cache,...

1d Array as 2d array

```
int* getElementAddr(int* arr, int r, int c) {
    return (arr+ r*COLS_NUM + c);
}

void foo(int* arr) {

   ... *(getElementAddr(arr,r,c))

}
```



1d Array as 2d array

```
int* getElementAddr(int* arr, int r, int c) {
    return (arr+ r*COLS_NUM + c);
}

void foo(int* arr) {

   ... *(getElementAddr(arr,r,c))

}
```

What can we do if we use a specific r and c many times and this is the bottleneck section?

1d Array as 2d array

```
void foo(int* arr) {

    ... size_t cache_index= r*COLS_NUM + c;
    ...
    ... *(arr + cache_index)

}
```

Use cache !

register

- **Register** - a small amount of **very fast memory**.
- Provides quick access to **commonly used values**.
- The register keyword specifies that the variable is to be stored in a machine register, **if possible.**
- Experience has shown that the **compiler** usually knows much better than humans **what** should go into registers **and when**.
- There is actually an **hierarchy of caches (L1, L2,…)**. We are not going to learn this in this course.

Volatile

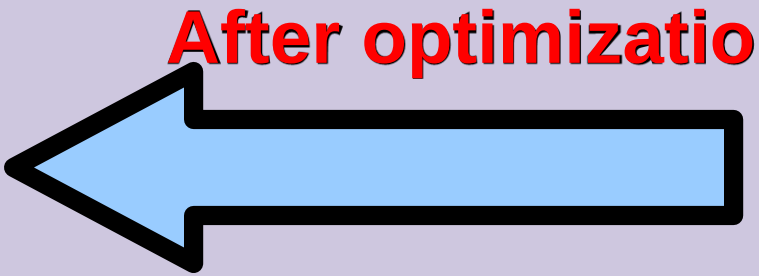
- variables that may be **modified externally** from the declaring object.
- Variables declared to be volatile will **not be optimized** by the compiler because their value can change at any time.

Volatile example

```
void foo(void)
{
    int* addr;
    addr= INPUT_ADDRESS;

    *addr = 0;

    while (*addr != 255)
        ;
}
```

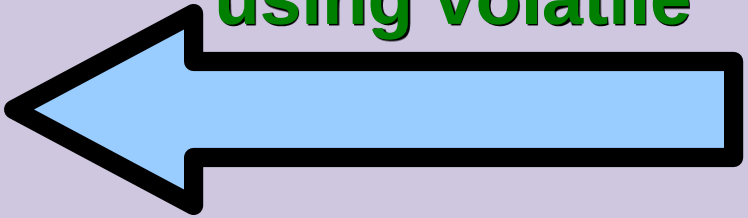
Volatile example

```
void foo(void)
{
    int* addr;
    addr= INPUT_ADDRESS;

    *addr = 0;

    while (1)
        ;
}
```



Volatile example

```
void foo(void)
{
    volatile int* addr;
    addr= INPUT_ADDRESS;

    *addr = 0;

    while (*addr != 255)
        ;
}
```



After optimization using volatile

union

`union` – a new type

A type that keeps (in different times) different types

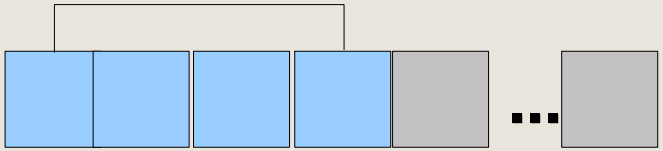
```
typedef union MyUnion
{
   int      i_val;
   double   d_val;
} MyUnion;
```

`union` – a new type, usage

```
typedef union MyUnion { int i_val; double d_val; } MyUnion;
```

```
MyUnion u;
```

```
u.i_val= 3;
printf("%d\n", u.i_val);
```



```
u.d_val= 3.22;
printf("%f\n", u.d_val);
```