C4GC

Dvir Yitzchaki

CEVA®

Enforcement

- Enforcement might be done by code review, by static analysis, by compiler, or by run-time checks.
- Wherever possible, we prefer “mechanical” checking (humans are slow, inaccurate, and bore easily) and static checking.

clang-tidy

- http://clang.llvm.org/extra/clang-tidy/checks/list.html
- Easy integration with CMake using:

  -DCMAKE_CXX_CLANG_TIDY:STRING="clang-tidy;-checks=cppcoreguidelines-*"

CppCoreCheck

- https://docs.microsoft.com/en-us/visualstudio/code-quality/using-the-cpp-core-guidelines-checkers
- Installed with VS2017
- Less easy integration with CMake:

```
-DCMAKE_CXX_FLAGS="/analyze /analyze:plugin
EspXEngine.dll"
```

- Needs environment variables:

```
set esp.extensions=cppcorecheck.dll
set esp.annotationbuildlevel=ignore
set caexcludepath=%include%
```

ES.20: Always initialize an object

```
int i;
if (val > 0) {
  i = 2;
}
return i;
```

ES.20: Always initialize an object

```
int i;
if (val > 0) {
  i = 2;
}
return i;
```

CppCoreCheck:

```
ninitialized1.cpp(12): warning C6001: Using uninitialized memory 'i'.:
Lines: 8, 9, 12
uninitialized1.cpp(8): warning C26494: Variable 'i' is uninitialized.
Always initialize an object (type.5:
http://go.microsoft.com/fwlink/p/?LinkID=620421).
```

ES.20: Always initialize an object

```
int i;
if (val > 0) {
  i = 2;
}
return i;
```

clang-tidy:

```
Uninitialized1.cpp:12:3: warning: Undefined or garbage value returned to caller [clang-analyzer-
core.uninitialized.UndefReturn]
  return i;
  ^
Uninitialized1.cpp:8:3: note: 'i' declared without an initial value
  int i;
  ^
Uninitialized1.cpp:9:7: note: Assuming 'val' is <= 0
  if (val > 0) {
      ^
Uninitialized1.cpp:9:3: note: Taking false branch
  if (val > 0) {
  ^
Uninitialized1.cpp:12:3: note: Undefined or garbage value returned to caller
  return i;
  ^
```

ES.20: Always initialize an object

```
auto i = 0;
if (val > 0) {
  i = 2;
}
return i;
```

ES.20: Always initialize an object

```
struct T1 {};
class T2 {
public:
  T2() {}
};
```

```
int n;

void foo() {
  int n2;
  std::string s;
  T1 t1;
  T2 t2;
}
```

ES.20: Always initialize an object

```
struct T1 {};
class T2 {
public:
  T2() {}
};
```

```
int n;

void foo() {
  int n2;
  std::string s;
  T1 t1;
  T2 t2;
}
```

CppCoreCheck:

```
uninitialized2.cpp(15): warning C26494: Variable 'n2' is uninitialized.
Always initialize an object (type.5:
http://go.microsoft.com/fwlink/p/?LinkID=620421).
uninitialized2.cpp(17): warning C26494: Variable 't1' is uninitialized.
Always initialize an object (type.5:
http://go.microsoft.com/fwlink/p/?LinkID=620421).
uninitialized2.cpp(18): warning C26496: The variable 't2' is assigned
only once, mark it as const (con.4:
https://go.microsoft.com/fwlink/p/?LinkID=784969).
```

ES.20: Always initialize an object

```
struct T1 {};
class T2 {
public:
  T2() {}
};
```

```
auto n = 0;

void foo() {
  auto n2 = 0;
  auto s = std::string{};
  auto t1 = T1{};
  auto t2 = T2{};
}
```

Con.4: Use const to define objects with values that do not change after construction

```
widget x{};
for (auto i = 2; i <= N; ++i) {
  x += some_obj.do_something_with(i);
}
```

Con.4: Use const to define objects with values that do not change after construction

```
widget x{};
for (auto i = 2; i <= N; ++i) {
  x += some_obj.do_something_with(i);
}
```

CppCoreCheck:

```
complexinit.cpp(19): warning C26496: The variable 'x' is assigned only
once, mark it as const (con.4:
https://go.microsoft.com/fwlink/p/?LinkID=784969).
```

ES.28: Use lambdas for complex initialization, especially of const variables

```
const widget x = [&] {
  widget val{};
  for (auto i = 2; i <= N; ++i) {
    val += some_obj.do_something_with(i);
  }
  return val;
}();
```

I.22: Avoid complex initialization of global objects

```
// in file GlobalInit1.cpp
extern char const *const term;
bool isVT100 = strcmp(term, "VT100") == 0;

// in file GlobalInit2.cpp
char const * const term = getenv("TERM");
```

I.22: Avoid complex initialization of global objects

```
// in file GlobalInit1.cpp
extern char const *const term;
bool isVT100 = strcmp(term, "VT100") == 0;

// in file GlobalInit2.cpp
char const * const term = getenv("TERM");
```

clang-tidy:

```
GlobalInit1.cpp:10:6: warning: initializing non-local variable with non-
const expression depending on uninitialized non-local variable 'term'
[cppcoreguidelines-interfaces-global-init]
bool isVT100 = strcmp(term, "VT100") == 0;
     ^
```

I.22: Avoid complex initialization of global objects

```
// in file GlobalInit1.cpp
extern char const *const term();
bool isVT100() {
  static bool is = strcmp(term(), "VT100") == 0;
  return is;
}

// in file GlobalInit2.cpp
char const *const term() {
  static char const *const theTerm = getenv("TERM");
  return theTerm;
}
```

C.48: Prefer in-class initializers to member initializers in constructors for constant initializers

```
class X {
  int i;
  std::string s;
  int j;

public:
  X() : i{666}, s{"qqq"} {}
  X(int ii) : i{ii} {}
};
```

clang-tidy:

```
MemberInit.cpp:20:3: warning: constructor does not initialize these fields: j [cppcoreguidelines-pro-type-member-init]
  X() : i{666}, s{"qqq"} {}
  ^
MemberInit.cpp:21:3: warning: constructor does not initialize these fields: j [cppcoreguidelines-pro-type-member-init]
  X(int ii) : i{ii} {}
  ^
```

C.48: Prefer in-class initializers to member initializers in constructors for constant initializers

```
class X {
  int i{666};
  std::string s{"qqq"};
  int j{0};

public:
  X() = default;
  X(int ii) : i{ii} {}
};
```

C.49: Prefer initialization to assignment in constructors

```
class B {
  std::string s1;

public:
  B(const std::string &name) { s1 = "Hello, " + name; }
};
```

C.49: Prefer initialization to assignment in constructors

```
class B {
  std::string s1;

public:
  B(const std::string &name) : s1{"Hello, " + name} {}
};
```

THANK YOU