# C Coding Standards

## Formatting

|  |  |
| --- | --- |
| 01 | Tabs that do not use ASCII spaces shall not be used |
| Description: | Example:  void foo()  {  int i; //Violation  }  Repair:  void foo()  {  int j; //OK  } |
| 02 | All curly braces "{ }" shall appear on a line by themselves |
| Description: | Example:  #define true 1  #define false 0  class C { /\* OK \*/  };  struct S { /\* Violation \*/  };  typedef enum E {e1, e2}; /\* OK \*/  void foo()  {  int bVal = 1;  int arr[] = {1, 2, 3}; // OK  while ( bVal )  { bVal = false; // Violation  }  }  Repair:  class C { /\* OK \*/  };  struct S  { /\* OK \*/  };  typedef enum E {e1, e2}; /\* OK \*/  void foo()  {  int bVal = 1;  int arr[] = {1, 2, 3}; // OK  while ( bVal )  {  bVal = false; // OK  }  } |
| 02 | All curly braces "{ }" shall appear on a line by themselves |
| Description: | Example:  #define true 1  #define false 0  class C {  } c; /\* OK \*/  struct S {  } s; /\* Violation \*/  typedef enum E {e1, e2}; /\* OK \*/  void foo()  {  int bVal = 1;  int arr[] = {1, 2, 3}; // OK  while ( bVal )  {  bVal = false; } // Violation  }  Repair:  #define true 1  #define false 0  class C {  } c; /\* OK \*/  struct S {  }; /\* OK \*/  struct S s;  typedef enum E {e1, e2}; /\* OK \*/  void foo()  {  int bVal = 1;  int arr[] = {1, 2, 3}; // OK  while ( bVal )  {  bVal = false; // OK  }  } |
| 03 | Physical lines should be less than eighty characters |
| Description: | Example:  int foo; // Violation  Repair:  int foo; // OK |
| 04 | The length of a macro should not exceed 10 lines |
| Description: | Example:  // Violation  #define SUM(\  PARAMETER\_1,\  PARAMETER\_2,\  PARAMETER\_3,\  PARAMETER\_4)\  (\  PARAMETER\_1+\  PARAMETER\_2+\  PARAMETER\_3+\  PARAMETER\_4\  )  void main()  {  int val = SUM(1,1,1,1);  } |
| 05 | Only one statement shall be allowed per line |
| Description: | Example:  void foo()  {  int i; char c; //Violation  }  Repair:  void foo()  {  int ii; // OK  char cc;  } |
| 06 | Logical parts of a conditional expression shall be grouped with parenthesis even if not logically required |
| Description: | Example:  void foo()  {  int x = 1;  int y = 1;  if (x == 0 || y == 0) //Violation  {  }  }  Repair:  void foo()  {  int x = 1;  int y = 1;  if ((x == 0) || (y == 0)) //OK  {  }  } |
| 06 | Logical parts of a conditional expression shall be grouped with parenthesis even if not logically required |
| Description: | Example:  void foo()  {  int x = 1;  int y = 1;  if (x == 0 || y == 0) // Violation  {  }  }  Repair:  void foo()  {  int x = 1;  int y = 1;  if ((x == 0) || (y == 0)) // OK  {  }  } |
| 07 | There shall be a single ASCII space character preceding and following assignment operators |
| Description: | Example:  int y = 1; //Violation  int z=1; //Violation  Repair:  int x = 1; //OK |
| 07 | There shall be a single ASCII space character preceding and following assignment operators |
| Description: | Example:  int y = 1; //Violation  int z=1; //Violation  Repair:  int x = 1; //OK |
| 08 | There shall be a single ASCII space character preceding and following bitwise operators |
| Description: | Example:  void foo()  {  int z = 73;  int y=0;  y=z|0x0f; //Violation  y=z | 0x0f; //Violation  }  Repair:  void foo()  {  int z = 73;  int y=0;  y=z | 0x0f; //OK  } |
| 08 | There shall be a single ASCII space character preceding and following bitwise operators |
| Description: | Example:  void foo()  {  int z = 73;  int y=0;  y=z|0x0f; //Violation  y=z | 0x0f; //Violation  }  Repair:  void foo()  {  int z = 73;  int y=0;  y=z | 0x0f; //OK  } |
| 09 | There shall be a single ASCII space character between a conditional statement and its opening parenthesis |
| Description: | Example:  void foo()  {  int x=1;  int y=1;    if(x == y) //Violation  {  }  }  Repair:  void foo()  {  int x=1;  int y=1;  if (x == y) //OK  {  }  } |
| 09 | There shall be a maximum of 1 ASCII space character following the opening parenthesis. |
| Description: | Example:  void foo()  {  int x=1;  int y=1;  if( x == y ) //Violation  {  }  }  Repair:  void foo()  {  int x=1;  int y=1;  if (x == y) //OK  {  }  if ( x == y ) //OK  {  }  } |
| 10 | There shall be a single ASCII space character preceding and following conditional operators |
| Description: | Example:  void foo()  {  int j=10;  int i=1;  j = ( i < 0 ) ? ( -i ) : ( i ); //Violation  j = ( i < 0 )?( -i ) : ( i ); //Violation  }  Repair:  void foo()  {  int j=10;  int i=1;  j = ( i < 0 ) ? ( -i ) : ( i ); //OK  } |
| 10 | There shall be a single ASCII space character preceding and following conditional operators |
| Description: | Example:  void foo()  {  int j=10;  int i=1;  j = ( i < 0 ) ? ( -i ) : ( i ); //Violation  j = ( i < 0 )?( -i ) : ( i ); //Violation  }  Repair:  void foo()  {  int j=10;  int i=1;  j = ( i < 0 ) ? ( -i ) : ( i ); //OK  } |
| 11 | There shall be a single ASCII space character preceding and following relational operator |
| Description: | Example:  void foo() {  int x = 1;  if(x==1) //Violation  {  }    if(x == 1) //Violation  {  }  if (x ==1) //Violation  {  }  }  Repair:  void foo() {  int x = 1;  if(x == 1) //OK  {  }  } |
| 12 | There shall be no white space preceding or following a primary operator |
| Description: | Example:  struct MyStruct {  int x;  };  void foo(struct MyStruct\* myObj) {  // ...  myObj-> x; //Violation  }  Repair:  struct MyStruct {  int x;  };  void foo(struct MyStruct\* myObj) {  // ...  myObj->x; //OK  } |
| 12 | There shall be no white space preceding or following a primary operator. |
| Description: | Example:  char tab2 [20]; //Violation  Repair:  char tab1[20]; //OK |
| 13 | There shall be a single ASCII space character following all commas |
| Description: | Example:  void foo()  {  int xx,yy; //Violation  int xxx, yyy; //Violation  }  Repair:  void foo()  {  int x, y; //OK  } |
| 14 | There shall be a single ASCII space character following all semicolons |
| Description: | Example:  void foo()  {  int i;  for ( i = 0;i <= 10;i++) //Violation  {  }  }  Repair:  void foo()  {  int i;  for ( i = 0; i <= 10; i++) //OK  {  }  } |
| 15 | There shall be no white space between an unary operator and its operand |
| Description: | Example:  void foo()  {  int i=0;  -- i; //Violation  }  Repair:  void foo()  {  int i=0;  --i; //OK  } |
| 16 | There shall be no white space between an unary operator and its operand |
| Description: | Example:  void foo()  {  int y = 0;  y ++; //Violation  }  Repair:  void foo()  {  int y = 0;  y++; //OK  } |
| 17 | There shall be no white spaces between the "return" or "sizeof " statements and its opening parenthesis |
| Description: | Example:  void foo()  {  sizeof (int); //Violation  }  Repair:  void foo()  {  sizeof(int); //OK  } |
| 17 | There shall be no white spaces surrounding "return" or "sizeof " statements argument or expression |
| Description: | Example:  void foo()  {  sizeof( int ); //Violation  }  Repair:  void foo()  {  sizeof(int); //OK  } |
| 17 | Parenthesis shall be used with the "return" and "sizeof" statements |
| Description: | Example:  #define true 1  #define false 0  int foo()  {  return true; //Violation  }  Repair:  #define true 1  #define false 0  int goo()  {  return(true); //OK  }  void xoo()  {  return; //OK  } |
| 18 | There shall be a single ASCII space character preceding and following logical operators |
| Description: | Example:  void foo()  {  int x = 1;  int y = 1;  if(x||y) //Violation  {  }  if(x || y) //Violation  {  }  }  Repair:  void foo()  {  int x = 1;  int y = 1;  if(x || y) //OK  {  }  } |

## Naming Conventions

|  |  |
| --- | --- |
| 01 | All "#define" constants shall be in uppercase |
| Description: | Example:  #define max\_value 255 // Violation  Repair:  #define MAX\_VALUE 255 // OK |
| 02 | In an enumerated list, list members (elements) shall be in uppercase and names or tags for the list shall be in lowercase. |
| Description: | Example:  enum Color { // Violation  red, // Violation  blue, // Violation  green // Violation  };  Repair:  enum color { // OK  RED, // OK  BLUE, // OK  GREEN // OK  };  References:  Motorola Coding Standards R-18 |
| 04 | Global prefixes should only be used for global variables. |
| Description: | Example:  void foo()  {  int theVar = 0; // Violation  } |
| 05 | Local variable names shall be lowercase. |
| Description: | Example:  void foo() {  int Count; // Violation  };  Repair:  void foo() {  int date; // OK  };  References:  Motorola Coding Standards R-20 |
| 06 | Begin all boolean type variables with 'b'. |
| Description: | Example:  void foo()  {  bool Var; // Violation  }  Repair:  void foo()  {  bool bVar; // OK  } |
| 07 | Begin class names with an uppercase letter. |
| Description: | Example:  class foo {}; // Violation  Repair:  class Foo {}; // OK |
| 08 | Avoid internal or external name conflict. |
| Description: | Example:  int bool = 0; // Violation  int catch = 0; // Violation  int class = 0; // Violation  int const\_cast = 0; // Violation  int delete = 0; // Violation  int dynamic\_cast = 0; // Violation  int explicit = 0; // Violation  int export = 0; // Violation  int false = 0; // Violation  int friend = 0; // Violation  int inline = 0; // Violation  int mutable = 0; // Violation  int namespace = 0; // Violation  int new = 0; // Violation  int operator = 0; // Violation  int private = 0; // Violation  int public = 0; // Violation  int protected = 0; // Violation  int reinterpret\_cast = 0; // Violation  int static\_cast = 0; // Violation  int template = 0; // Violation  int this = 0; // Violation  int throw = 0; // Violation  int true = 0; // Violation  int try = 0; // Violation  int typeid = 0; // Violation  int typename = 0; // Violation  int using = 0; // Violation  int virtual = 0; // Violation  int wchar\_t = 0; // Violation |
| 09 | Begin constant variables with 'c'. |
| Description: | Example:  const int Foo = 0; // Violation  Repair:  const int cFoo = 0; // OK |
| 10 | Begin class data member names with 'its'. |
| Description: | Example:  class Foo {  private:  int bar; // Violation  };  Repair:  class Foo {  private:  int itsBar; // OK  }; |
| 11 | Begin all double type variable with 'd'. |
| Description: | Example:  void foo() {  double Var; // Violation  }  Repair:  void foo() {  double dVar; // OK  } |
| 12 | Enumerated type name shall begin with an uppercase letter prefixed by software element and suffixed by '\_t'. |
| Description: | Example:  enum LOC\_PossibleColors { RED2, BLUE2, GREEN2 }; // Violation  Repair:  enum LOC\_PossibleColors\_t { RED, BLUE, GREEN }; // OK |
| 13 | Begin all float type variables with 'f'. |
| Description: | Example:  void foo()  {  float Var; // Violation  }  Repair:  void foo()  {  float fVar; // OK  } |
| 14 | Begin all function names with uppercase letter. |
| Description: | Example:  void foo(); // Violation  Repair:  void Foo(); // OK |
| 15 | Begin global variable names with 'the'. |
| Description: | Example:  int global\_var; // Violation  Repair:  int theWindows; // OK |
| 16 | Begin all integer type variable with 'i'. |
| Description: | Example:  void foo()  {  int Var; // Violation  }  Repair:  void foo()  {  int iVar; // OK  } |
| 17 | 'is' functions should return bool values. |
| Description: | Example:  int isPos(int x) // Violation  {  return x > 0;  }  Repair:  bool isPositive(int x) // OK  {  return x > 0;  } |
| 18 | Begin all long integer variables with 'li'. |
| Description: | Example:  void foo()  {  long int Var; // Violation  }  Repair:  void foo()  {  long int liVar; // OK  } |
| 19 | Prefix a variable type 'pointer' with a 'p' character |
| Description: | Example:  int \*Foo; // Violation  Repair:  int \*pFoo; // OK |
| 20 | Begin all short integer variables with 'si'. |
| Description: | Example:  void foo()  {  short int Var; // Violation  }  Repair:  void foo()  {  short int siVar; // OK  } |
| 21 | Begin all signed character variables with 'c'. |
| Description: | Example:  void foo()  {  signed char Var; // Violation  }  Repair:  void foo()  {  signed char cVar; // OK  } |
| 22 | Begin all terminated characters string variables with 'sz'. |
| Description: | Example:  void foo()  {  char Var[] = "\n"; // Violation  }  Repair:  void foo()  {  char szVar[] = "\n"; // OK  } |
| 23 | Struct type name shall begin with an uppercase letter prefixed by software element and suffixed by '\_t' |
| Description: | Example:  struct LOC\_Position {}; // Violation  Repair:  struct LOC\_Position\_t {}; // OK |
| 24 | Begin all unsigned character type variables with 'uc'. |
| Description: | Example:  void foo()  {  unsigned char Var; // Violation  }  Repair:  void foo()  {  unsigned char ucVar; // OK  } |
| 25 | Begin all unsigned integer type variables with 'ui'. |
| Description: | Example:  void foo()  {  unsigned int Var; // Violation  }  Repair:  void foo()  {  unsigned int uiVar; // OK  } |
| 26 | Begin all variable with a lowercase letter. |
| Description: | Example:  int Global\_2; // Violation  class Foo  {  private:  int \_i; // Violation  float F; // Violation  };  void foo()  {  int Local\_var; // Violation  }  Repair:  int global\_1; // OK  class Foo  {  private:  int j; // OK  };  void foo()  {  int local; // OK  } |
| 27 | Use lowercase letters for structure and union member names. |
| Description: | Example:  struct date\_foo2 {  int datemonth;  int DateDay; // Violation  int Dateyear; // Violation  };  Repair:  struct date\_foo1 {  int dateMonth;  int dateDay; // OK  int dateYear; // OK  };  References:  Motorola Coding Standards R-19 |
| 28 | Append names of non-scalar typedefs with "\_t". |
| Description: | Example:  class MyClass {  int count;  };  typedef MyClass My; // Violation  Repair:  class MyClass {  int count;  };  typedef MyClass My\_t; // OK  References.  Motorola Coding Standards R-17 |

## Metric

|  |  |
| --- | --- |
| 18 | Follow the limit for Cyclomatic Complexity |
| Description: | Example:  void foo(int a, int b) // Violation  {  switch (a) // 1  {  case 1: //2  break;  case 2: //3  break;  case 3: //4  break;  case 4: //5  break;  case 5: //6  break;  case 6: //7  break;  case 7: //8  break;  case 8: //9  break;  case 9: //10  break;  case 10: //11  break;  case 11: //12  break;  case 12: //13  break;  case 13: //14  break;  case 14: //15  break;  case 15: //16  break;  }  if(a||b) //17,18  {  }  if(a&&b) //19,20  {  }  if((a||b)&&(a&&b)) // 21,22,23,24  {  }  do //25  {  }while(1==1);  if(1==1){} //26  if(1==1){} //27  if(1==1){} //28  if(1==1){} //29  if(1==1){} //30  }  Repair:  Keep a cyclomatic complexity value on the level lower than 30.  References:  Robert B. Grady: "Practical Software Metrics For Project Management and  Process Improvement" Prentice Hall P T R, 1992. pp.16 - 18 |
| 01 | Avoid functions with over 50 lines |
| Description: | Example:  void too\_long\_function() { // Violation  int a1;  int a2;  int a3;  int a4;  int a5;  int a6;  int a7;  int a8;  int a9;  int a10;  int a11;  int a12;  int a13;  int a14;  int a15;  int a16;  int a17;  int a18;  int a19;  int a20;  int a21;  int a22;  int a23;  int a24;  int a25;  int a26;  int a27;  int a28;  int a29;  int a30;  int a31;  int a32;  int a33;  int a34;  int a35;  int a36;  int a37;  int a38;  int a39;  int a40;  int a41;  int a42;  int a43;  int a44;  int a45;  int a46;  int a47;  int a48;  int a49;  int a50;  int a51;  int a52;  int a53;  int a54;  }  Repair:  Limit the size of the function to 50 lines. |
| 02 | Avoid switch statements with many cases |
| Description: | Example:  void foo(int i) {  // There are over 10 cases in the switch statement.  switch (i) { // Violation  case 1:  break;  case 2:  break;  case 3:  break;  case 4:  break;  case 5:  break;  case 6:  break;  case 7:  break;  case 8:  break;  case 9:  break;  case 10:  break;  case 11:  break;  default:  break;  }  }  Repair:  Look for cleaner ways to invoke the alternative behaviors. |
| 03 | Number of blocks of code in a function |
| Description: | Example:  void func(int i) { // Violation  if (1) {  // do something;  } else {  // do another;  }  if (1) {  // do something;  } else {  // do another;  }  if (1) {  // do something;  } else {  // do another;  }  if (1) {  // do something;  } else {  // do another;  }    if (1) {  // do something;  }  do {  } while(1);  switch(i) {  case 1:  break;  default:  break;  }  }  Repair:  The number of blocks within the function should be limited to 10.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, November/December 1999, pg. 34. |
| 04 | Number of function calls within function |
| Description: | Example:  class Foo {  public:  void func1() {} // OK  void func2() { // Violation  func1();  func1();  func1();  func1();  func1();  func1();  func1();  func1();  func1();  func1();  func1();  }  };  Repair:  Limit the number of function calls within the function to 10.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, Novemeber/December, 1999, pg. 32. |
| 05 | Class inheritance level |
| Description: | Example:  class Base {}; // OK  class Dev1 : public Base {}; // OK  class Dev2 : public Dev1 {}; // OK  class Dev3 : public Dev2 {}; // OK  class Dev4 : public Dev3 {}; // OK  class Dev5 : public Dev4 {}; // OK  class Dev6 : public Dev5 {}; // OK  class Dev7 : public Dev6 {}; // OK  class Dev8 : public Dev7 {}; // OK  class Dev9 : public Dev8 {}; // OK  class Dev10 : public Dev9 {}; // OK  class Dev11 : public Dev10 {}; // Violation  Repair:  Limit he number of base classes to 10.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, Novemeber/December, 1999, pg. 33. |
| 06 | Number of data member(s) per class |
| Description: | Example:  class Foo { // Violation  public:  int i1;  int i2;  int i3;  int i4;  int i5;  int i6;  protected:  int j1;  int j2;  int j3;  int j4;  int j5;  int j6;  int j7;  private:  int k1;  int k2;  int k3;  };  Repair:  Limit the number of data members to 15 per class.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, Novemeber/December, 1999, pg. 33. |
| 07 | Number of methods per class |
| Description: | Example:  class Foo { // Violation  public:  void method1();  void method2();  void method3();  void method4();  void method5();  void method6();  protected:  void method7();  void method8();  void method9();  void method10();  void method11();  void method12();  private:  void method13();  void method14();  void method15();  void method16();  void method17();  void method18();  void method19();  void method20();  void method21();  };  Repair:  Limit the number of methods to 20 per class.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, Novemeber/December, 1999, pg. 33. |
| 08 | Number of parameter(s) per method |
| Description: | Example:  class Foo {  public:  void func(int i, double d, float f, char c, Foo &); // OK  void func2(int ,int ,int ,int ,int ,int ,int ,int ,int ,int ,int ); // Violation  };  Repair:  Limit the number of parameters to 10 per method.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, Novemeber/December, 1999, pg. 34. |
| 09 | Number of private data member(s) per class |
| Description: | Example:  class Foo { // Violation  private:  int i1;  int i2;  int i3;  int i4;  int i5;  int i6;  int j1;  int j2;  int j3;  int j4;  int j5;  int j6;  int j7;  int k1;  int k2;  int k3;  };  Repair:  Limit the number of private data members to 15 per class.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, November/December 1999, pg. 33. |
| 10 | Number of private methods per class |
| Description: | Example:  class Foo { // Violation  private:  void method1();  void method2();  void method3();  void method4();  void method5();  void method6();  void method7();  void method8();  void method9();  void method10();  void method11();  };  Repair:  Limit the number of private methods to 10 per class.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, November/December 1999, pg. 33. |
| 11 | Number of protected data member(s) per class |
| Description: | Example:  class Foo { // Violation  protected:  int i1;  int i2;  int i3;  int i4;  int i5;  int i6;  int j1;  int j2;  int j3;  int j4;  int j5;  int j6;  int j7;  int k1;  int k2;  int k3;  };  Repair:  Limit the number of protected data members to 15 per class.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, November/December 1999, pg. 33. |
| 12 | Number of protected methods per class |
| Description: | Example:  class Foo { // Violation  protected:  void method1();  void method2();  void method3();  void method4();  void method5();  void method6();  void method7();  void method8();  void method9();  void method10();  void method11();  };  Repair:  Limit the number of protected methods to 10 per class.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, November/December 1999, pg. 33. |
| 13 | Number of public data member(s) per class |
| Description: | Example:  class Foo { // Violation  public:  int i1;  int i2;  int i3;  int i4;  int i5;  int i6;  int j1;  int j2;  int j3;  int j4;  int j5;  int j6;  int j7;  int k1;  int k2;  int k3;  };  Repair:  Limit the number of public data members to 15 per class.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, November/December 1999, pg. 33. |
| 14 | Number of public methods per class |
| Description: | Example:  class Foo { // Violation  public:  void method1();  void method2();  void method3();  void method4();  void method5();  void method6();  void method7();  void method8();  void method9();  void method10();  void method11();  void method12();  void method13();  void method14();  void method15();  void method16();  void method17();  void method18();  void method19();  void method20();  void method21();  };  Repair:  Limit the number of public methods to 20 per class.  References:  By Mark Schroeder "A Practical Guide to Object-Oriented Metrics"  IT Professional, November/December 1999, pg. 33. |
| 15 | Avoid functions with more than 5 parameters |
| Description: | Example:  void foo(int a, int b, int c, int d, int e, int f) // Violation  {  }  Repair:  Limit the number of parameters to 5 per function.  References:  Motorola Coding Standards G-6 |
| 16 | Macros should not use more than 5 parameters |
| Description: | Example:  #define PLUS(a,b,c,d,e,f) (a+b+c+d+e+f) // Violation  Repair:  Limit the number of parameters to 5 per macro. |
| 17 | Avoid structs, unions, or classes with more than 20 fields |
| Description: | Example:  struct \_tA // Violation  {  int i0;  int i1;  int i2;  int i3;  int i4;  int i5;  int i6;  int i7;  int i8;  int i9;  int i10;  int i11;  int i12;  int i13;  int i14;  int i15;  int i16;  int i17;  int i18;  int i19;  int i20;  int i21;  int i22;  int i23;  };  Repair:  Limit the number of fields to 20. |
| 19 | The percentage of comment lines versus the total number of module lines should be between 20 and 60 |
| Description: | Example:  void foo1(int y) { // Violation  //  //  //  //  }  void foo12() { // Violation  /\*  \*/  }  void foo3() { // Violation  /\*\*//\*\*//\*\*//\*\*/  }  void foo4() { // Violation  }  Repair:  void foo1() { // OK  //  //  //  //  }  void foo2() { // OK  /\*  \*/  }  void foo3() { // OK  /\*\*//\*\*//\*\*//\*\*/  } |

## Coding Conventions

|  |  |
| --- | --- |
| 01 | Array elements shall be accessed by the array operator [ ] |
| Description: | Example:  void foo()  {  int array[2];  \*(array+1) = 0; //Violation  \*array = 0; //Violation  }  Repair:  void foo()  {  int array[2];  array[1] = 0; //OK  }  References:  Motorola Coding Standards R-46 |
| 02 | Ellipses shall not be used |
| Description: | Example:  class A  {  public:  int foo( int a, ... ); // Violation  };  Repair:  class A  {  public:  int foo( int a, int b = 0 ); // OK  }; |
| 03 | Member functions shall not be defined within the class definition |
| Description: | Example:  class A // Violation  {  public:  int foo() {};  inline int bar();  };  inline int A::bar() {  return (0) ;  };  Repair:  class A // OK  {  public:  int foo();  inline int bar();  };  int A::foo() {};  inline int A::bar() {  return (0) ;  }; |
| 03 | Member functions shall not be defined within the class definition |
| Description: | Example:  template <class T>  class A {  public:  void foo(){} // Violation  };  Repair:  template <class T>  class A {  public:  void foo();  };  template <class T> void B<T>::foo(){} // OK |
| 04 | Do not use the goto statement |
| Description: | Example:  int main( ) {  goto end; // Violation  end:  return 0;  } |
| 05 | Do not define any constructor or destructor to be inline |
| Description: | Example:  class A  {  public:  A( ) {} // Violation  ~A( ) {} // Violation  };  Repair:  class A  {  public:  A( ); // OK  ~A( ); // OK  };  A::A (){};  A::~A (){}; |
| 06 | Do not define structs that contain member functions |
| Description: | Example:  struct A  {  public:  int foo(); // Violation  };  Repair:  struct A // OK  {  }; |
| 07 | Avoid using the ?: operator |
| Description: | Example:  int main( )  {  return (1 ? 1 : 0); // Violation  };  Repair:  int main( ) {  if (1)  return 1;  else  return 0;  }; |
| 08 | If a function has no parameters, use ( ) instead of ( void ) |
| Description: | Example:  void foo( void ) // Violation  {  int \* d;  void\* p = (void \*) d;  }  Repair:  void foo( ) // OK  {  int \* d;  void\* p = (void \*) d;  } |
| 09 | Do not use break in for loops |
| Description: | Example:  void func()  {  for (int j = 0; j < 10; j++) {  if (j==5)  {  break; // Violation  }  break; // Violation  }  }  Repair:  void func()  {  for (int j = 0; j < 10; j++) {  while (true)  {  break; // OK - 'break' in while context  }  }  } |
| 10 | Do not type cast pointers to primitive types for variables that have been declared as pointers to functions |
| Description: | Example:  void Foo(char \*ptrC)  {  \*ptrC = 0;  return;  }  void f()  {  void \*ptrV = 0;  void (\*funPtr) (char\*) = 0;  funPtr = &Foo;  ptrV = (void\*)funPtr; // Violation  return;  } |
| 11 | The comma operator shall only be used in for statements and variable declarations |
| Description: | Example:  void foo(int i)  {  int dVal, count;  foo((dVal=i,count=dVal+1)); // Violation  }  Repair:  void foo(int i)  {  int dVal,count; // OK  }  References:  Motorola Coding Standards R-49 |
| 12 | All condition statements shall use '{' and '}' to identify the body of code associated with the condition |
| Description: | Example:  void foo(char \*str)  {  if ((\*str) != 0) // Violation  str++;  }  Repair:  void foo(char \*str)  {  if ((\*str) != 0) // OK  {  str++;  }  } |
| 13 | All condition statements shall use '{' and '}' to identify the body of code associated with the condition |
| Description: | Example:  void foo(char \*str)  {  do // Violation  str++;  while(\*str!=0);  }  Repair:  void foo(char \*str)  {  do // OK  {  str++;  }  while(\*str!=0);  } |
| 14 | All condition statements shall use '{' and '}' to identify the body of code associated with the condition |
| Description: | Example:  void foo(char \*str)  {  while ((\*str) != 0) // Violation  str++;  }  Repair:  void foo(char \*str)  {  while ((\*str) != 0) // OK  {  str++;  }  } |
| 15 | Declare reference parameters as const references whenever possible |
| Description: | Example:  struct Foo  {  int x;  int y;  };  int Bar(Foo &f) // Violation  {  return f.x;  }  int FooBar(Foo &f) // OK  {  return f.x++;  }  Repair:  struct Foo  {  int x;  int y;  };  int Bar(const Foo &f) // OK  {  return f.x;  } |
| 16 | Pass const pointer to function if the value of which the pointer points at is not to be changed |
| Description: | Example:  int foo( int \*j) // Violation  {  return \*j;  }  Repair:  int bar( const int \*j) //OK  {  return \*j;  }  References:  Motorola Coding Standards G-10 |
| 17 | Do not declare the size of an array when the array is passed into a function as a parameter |
| Description: | Example:  void foo2(int ii[30]) // Violation  {  }  void foo3(char a,int ii[30][30][30]) // Violation  {  }  Repair:  void foo1(int ii[]) //OK  {  }  void foo4(char a,int ii[][30][30]) //OK  {  }  References:  Motorola Coding Standards R-7 |
| 18 | Do not declare the size of an array when the array is initialized |
| Description: | Example:  #define SIZE 4  int tab1[SIZE] = {1,2,3}; // Violation  Repair:  int tab2[]={1,2,3}; // OK  References:  Motorola Coding Standards, R-6 |
| 19 | Always provide a default branch for switch statements |
| Description: | Example:  int main()  {  int a = 0;  switch (a) // Violation  {  case 0:  return 0;  case 1:  return 1;  }  }  Repair:  int main()  {  int a = 0;  switch (a) // OK  {  case 0:  return 0;  case 1:  return 1;  default:  return 0;  }  } |
| 20 | Prefer while statements over do statements |
| Description: | Example:  void func( )  {  int i = 0;  do // Violation  {  i++;  } while ( i < 10);  }  Repair:  void func( )  {  int i = 0;  while (i < 10) // OK  {  i++;  }  } |
| 21 | Do not use the 'enum' keyword to declare a variable in C++. |
| Description: | Example:  enum Colors { RED, BLUE, GREEN };  enum Colors c; // Violation  Repair:  enum Colors { RED, BLUE, GREEN };  Colors c; // OK |
| 22 | EOS should be used to terminate a string rather than NULL |
| Description: | Example:  #define EOS '\0'  void foo()  {  char str[30] = "Sample text.";  str[7] = NULL; // Violation  }  Repair:  #define EOS '\0'  void foo()  {  char str[30] = "Sample text.";  str[7] = EOS; // OK  } |
| 23 | When using enum, the values of each member should be explicitly declared |
| Description: | Example:  enum my\_enum1  {  a, // Violation  };  Repair:  enum my\_enum2  {  a = 0 // OK  };  References:  Motorola Coding Standards G-15 |
| 24 | An explicit logical comparison should be used in conditional expression |
| Description: | Example:  void foo()  {  int isvisible = 1;  if (isvisible) // Violation  {  ;  }    if (!isvisible) // Violation  {  ;  }  }  Repair:  void foo()  {  int isvisible = 1;  if (isvisible == 1) // OK  {  ;  }    if (isvisible != 1) // OK  {  ;  }  } |
| 25 | Functions shall explicitly declare their return type |
| Description: | Example:  foo( ); // Violation  Repair:  int foo( ); // OK |
| 26 | Avoid passing expressions like assignment to sizeof() operators |
| Description: | Example:  void foo ()  {  int iVar1 = 0;  int iVar2 = 0;  iVar1 = sizeof(iVar2 = 1); // Violation  return;  } |
| 27 | If FALSE is to be defined, and is not already defined, FALSE should be 0 |
| Description: | Example:  #define FALSE 1 // Violation  #undef FALSE  #define FALSE // Violation  Repair:  #define FALSE 0 // OK |
| 28 | If FALSE is to be defined, and is not already defined, FALSE should be 0 |
| Description: | Example:  #ifndef FALSE  typedef enum  {  TRUE = 1,  FALSE = -1 // Violation  } BOOL ;  #endif |
| 29 | Don't assign to loop control variables in the body of a for loop |
| Description: | Example:  void func()  {  for (int i = 0; i < 100; i++)  {  i += 3; // Violation  }  for (int j = 0; j < 100; j++)  {  j++; // Violation  }  }  Repair:  Remove assignments to loop control variable from body of the for statement. |
| 30 | Avoid functions that modify the global variable |
| Description: | Example:  int theGlob;  void foo( int iVar )  {  theGlob = -1; // Violation  if (iVar == 0 )  {  theGlob = 0; // Violation  } else {  theGlob = 1; // Violation  }  return;  } |
| 31 | Avoid global variables |
| Description: | Example:  int theVar; //Violation  void foo(int iVar)  {  theVar = iVar;  return;  } |
| 32 | "#define" or enum constants should be used instead of hard coded values whenever possible |
| Description: | Example:  void foo()  {  int tabColors[256]; // Violation  if ( tabColors[2] == 1 ) // Violation  {  /\*...\*/  }  }  Repair:  #define buff 256  #define OK 1  enum color  {  RED = 0,  BLUE = 1,  GREEN = 2  /\*...\*/  };  void foo()  {  int tabColors[buff]; // OK  if ( tabColors[RED] == OK ) // OK  {  /\*...\*/  }  } |
| 34 | If, else, while and do statements shall be followed by a block, even if it is empty |
| Description: | Example:  void foo() {  int x;  if (1) // Violation  if (1) // Violation  if (1)  {  x=0;  }    if (1) // Violation  ;  while(1) // Violation  x++;  }  Repair:  void foo() {  int x;  if (1) // OK  {x=0;}  if (1) // OK  {  x=0;  }  if (1) { // OK  if (1)  {  x=0;  }  }  if (1) { // OK  ;  }  while(1) { // OK  x++;  }  } |
| 35 | Don't use macros in include statement |
| Description: | Example:  /\* Notice:  \* You must be sure that your compiler can do that  \* and that given header file is available  \* Works with gcc  \*/  #define A(string) #string  #define HEADER\_FILE(nr) A(MacroWithinInclude ## nr ##.h)  #include HEADER\_FILE(12) // Violation  #include "MacroWithinInclude12.h" // OK |
| 36 | Do not use operator ++ or -- in the conditional expression of if, while, or switch |
| Description: | Example:  int Foo (int iVar)  {  if (iVar-- && iVar<10) // Violation  {  iVar += 10;  }  else  {  iVar -= 10;  }  return iVar;  } |
| 37 | Incorrect NULL definition |
| Description: | Example:  #define NULL 0 // Violation  #undef NULL  #define NULL // Violation  Repair:  #define NULL (void\*)0 // OK |
| 38 | Provide only one return statement in a function |
| Description: | Example:  int main()  {  if (0) {  return 0;  } else if (1) {  return 1; // Violation  } else {  return 2; // Violation  }  }  Repair:  int main()  {  int result = 0;  if (0) {  result = 0;  } else if (1) {  result = 1;  } else {  result = 2;  }  return result; // OK  } |
| 38 | Provide only one return statement in a function |
| Description: | Example:  class Sample  {  int foo();  };  int Sample::foo()  {  if (0) {  return 0;  } else if (1) {  return 1; // Violation  } else {  return 2; // Violation  }  }  Repair:  class Sample  {  int foo();  };  int Sample::foo()  {  int result = 0;  if (0) {  result = 0;  } else if (1) {  result = 1;  } else {  result = 2;  }  return result; // OK  } |
| 39 | Don't define part of statement |
| Description: | Example:  #define PARTIAL(a) ((a) \* // Violation |
| 40 | Pass built-in-types by value unless you are modifying them |
| Description: | Example:  int Foo(int i, int &j) // Violation  {  return i + j;  }  int Bar(int &i, int j) // Violation  {  j += i;  return j;  }  Repair:  int Foo(int i, int j) // OK  {  return i + j;  }  int Bar(int i, int j) // OK  {  j += i;  return j;  } |
| 41 | Don't redefine control statements |
| Description: | Example:  #define if while // Violation  #define for if // Violation  #define else try //Violation |
| 42 | Don't redefine primitive types |
| Description: | Example:  #define T30 char //Violation  #define T31 unsigned char // OK |
| 43 | Don't redefine primitive types. Use typedef |
| Description: | Example:  #define INT int // Violation |
| 44 | Separate logical tests in conditional expressions |
| Description: | Example:  void foo( int j, int i) {  if (i != j > 0) {} // Violation  }  Repair:  void foo( int j, int i) {  if (i != j && j > 0) {} // OK  }  References:  Motorola Coding Standards G-12 |
| 45 | Avoid switch statements with only one case |
| Description: | Example:  void foo ()  {  int i = 0;  switch(i) // Violation  {  case 0: break;  default: break;  }  return;  }  Repair:  void foo ()  {  int i = 0;  if(i==0) // OK  { }  else { }  return;  } |
| 47 | Do not use the 'struct' keyword to declare a variable in C++ |
| Description: | Example:  struct Position\_t {};  struct Position\_t Pos; // Violation  Repair:  struct Position\_t {};  Position\_t Pos; // OK |
| 48 | If TRUE is to be defined, and is not already defined, TRUE should be 1 |
| Description: | Example:  #define TRUE 0 //Violation  #undef TRUE  #define TRUE //Violation  #undef TRUE  Repair:  #define TRUE 1 //OK |
| 49 | If TRUE is to be defined, and is not already defined, TRUE should be 1 |
| Description: | Example:  #ifndef TRUE  typedef enum  {  FALSE = 0,  TRUE = 3 // Violation  } BOOL ;  #endif  Repair:  #ifndef TRUE  typedef enum  {  FALSE = 0,  TRUE = 1 // OK  } BOOL ;  #endif |
| 50 | Storage type modifiers shall be associated with the type, not the variable |
| Description: | Example:  int static i; //Violation  Repair:  static char j; //OK |
| 51 | Remove unnecessary '== true' in Boolean functions |
| Description: | Example:  bool isPositive(int a);  bool foo()  {  return isPositive(5) == true; // Violation  }  Repair:  bool isPositive(int a);  bool foo1()  {  return isPositive(5); // OK  } |
| 52 | In a function such as macros, before and after multiplication or division, use parentheses |
| Description: | Example:  #define DIVISION\_1(x) (x / (x)) //Violation  #define DIVISION\_2(x) ((x) / x) //Violation  #define DIVISION\_3(x) (x / (x)) //Violation  #define MULTI\_1(x) (x \* (x)) //Violation  #define MULTI\_2(x) ((x) \* x) //Violation  #define MULTI\_3(x) (x \* (x)) //Violation  Repair:  #define DIVISION\_1(x) ((x) / (x)) //OK  #define MULTI\_3(x) ((x) \* (x)) //OK |
| 53 | Use positive logic rather than negative logic whenever practical |
| Description: | Example:  void foo( int \*j) {  if(j!=0){ //Violation  (\*j)++;  }  }  Repair:  void foo( int \*j) {  if(j==0){  } else { //OK  (\*j)++;  }  }  References:  Motorola Coding Standards G-13 |
| 54 | Pointers to functions shall be implemented with a typedef |
| Description: | Example:  void (\*foo) (int); // Violation  Repair:  typedef void (\*bar)(int);  bar foo; // OK |
| 55 | All structures should have typedefs |
| Description: | Example:  struct A // Violation - no typedef  {  int i;  };  Repair:  typedef struct A // OK  {  int i;  } A\_t;  or  struct A // OK  {  int i;  };  typedef struct A A\_t;  Reference:  Customer guidelines |
| 56 | void should be used when a function is passed or returns no values |
| Description: | Example:  func1 () { //VIOLATION, no return type, no void in parameter list  printf("%s\n", "Hello World");  };  void func2() { //VIOLATION, no void parameter list  int a = 2000;  /\* do something \*/  };  func3 (void) { //VIOLATION, no void as explicit return type  return ;  };  Repair:  void func1 (void) {  printf("%s\n", "Hello World");  }  void func2 (void) {  int a = 2000;  /\* do something \*/  }  void func3 (void) {  return ;  };  Reference:  Customer guidelines |
| 57 | .c files may not include other .c files |
| Description: | Example:  //file.c  #include "file2.c" // Violation  Repair:  1. Compile each .c file separately and link them.  2. Develop a convention for naming implementation files  to be included that does not use .c or .C extensions.  Reference:  Customer guidelines. |
| 58 | Provide error handling for file opening errors right next to the call to fopen |
| Description: | Example:  void foo()  {    FILE\* data = fopen("data.txt", "r"); // Violation  cout<<data;  FILE\* data2;  data2 = fopen("data.txt", "r"); // Violation  cout<<data2;    if (!data2){ }  }  Repair:  void foo()  {    FILE\* data = fopen("data.txt", "r"); // OK  if (!data)  {  cout<<"File opening error";  }  cout<<data;  FILE\* data2;  if (data2 = fopen("data.txt", "r")) // OK  {  cout<<data2;  }  } |
| 59 | Name conflicts between variable names and type names |
| Description: | Example:  struct Name{  };  static void raw\_Add(int Name, struct Name\* RHS); // Violation  Repair:  struct Name{  };  static void raw\_Add(int AuthorName, struct Name\* RHS); // OK |
| 60 | Conversion operator, operator->, operator(), operator[] should be const |
| Description: | Example:  class A {  public:  A& operator()( int x ); // Violation  A& operator[]( int x ); // Violation  A& operator->( int x ); // Violation  operator int( ); // Violation  };  Repair:  class A {  public:  A& operator()( int x ) const; // OK  A& operator[]( int x ) const; // OK  A& operator->( int x ) const; // OK  operator int( ) const; // OK  }; |
| 61 | Bitwise operators, comparison operators, logical operators, comma operator should be const |
| Description: | Example:  class A {  public:  A& operator^( int x ); // Violation  A& operator==( int x ); // Violation  A& operator&&( int x ); // Violation  A& operator,( int x ); // Violation  };  Repair:  class A\_ {  public:  A& operator^( int x ) const; // OK  A& operator==( int x ) const; // OK  A& operator&&( int x ) const; // OK  A& operator,( int x ) const; // OK  }; |
| 62 | Local variables/parameters and class/parent class/parent struct member variables should have different name. |
| Description: | Example:  class Base {  void foo( ) {  float memberF; // Violation  }  int foo1( );  short foo2( ) {  for( int \_var2 = 0; \_var2 < 10; \_var2++ ) { // Violation  }  return Var3;  }  public:  int MemberF;  float \_Var1;  float \_Var2;  short Var3;  };  int Base::foo1( ) {  int \_var1 = 6; // Violation  return \_var1;  }  class A : public Base {  void foo( ) {  float memberF; // Violation  }  int foo1( );  short foo2( ) {  for( int \_var2 = 0; \_var2 < 10; \_var2++ ) { // Violation  }  return Var3;  }  };  int A::foo1( ) {  int \_var1 = 6; // Violation  return \_var1;  }  Repair:  class Base {  void foo( ) {  float fLocal; // OK  }  int foo1( );  int x;  short foo2( ) {  int y; // OK  for( int iValue = 0; iValue < 10; iValue++ ) { // OK  }  return Var3;  }  public:  int MemberF;  float Var1;  float \_Var2;  short Var3;  };  int Base::foo1( ) {  int iValue = 6; // OK  return iValue;  }  class A : public Base {  void foo( ) {  float fLocal; // OK  }  int foo1( );  int x;  short foo2( ) {  int y; // OK  for( int iValue = 0; iValue < 10; iValue++ ) { // OK  }  return Var3;  }  };  int A::foo1( ) {  int iValue = 6; // OK  return iValue;  } |
| 63 | Local variable or parameter names and class member variable or parent class/struct member variable names shall differ by more than a single character. |
| Description: | Example:  class Base {  void foo( ) {  float memberF; // Violation  }  int foo1( );  short foo2( ) {  for( int \_var2 = 0; \_var2 < 10; \_var2++ ) { // Violation  }  return Var3;  }  public:  int Member;  float Var1;  float Var2;  short Var3;  };  int Base::foo1( ) {  int \_var1 = 6; // Violation  return \_var1;  }  class A : public Base {  void foo( ) {  float memberF; // Violation  }  int foo1( );  short foo2( ) {  for( int \_var2 = 0; \_var2 < 10; \_var2++ ) { // Violation  }  return Var3;  }  };  int A::foo1( ) {  int \_var1 = 6; // Violation  return \_var1;  }  Repair:  class Base {  void foo( ) {  float fLocal; // OK  }  int foo1( );  int x;  short foo2( ) {  int y; // OK  for( int iValue = 0; iValue < 10; iValue++ ) { // OK  }  return Var3;  }  public:  int Member;  float Var1;  float Var2;  short Var3;  };  int Base::foo1( ) {  int iValue = 6; // OK  return iValue;  }  class A : public Base {  void foo( ) {  float fLocal; // OK  }  int foo1( );  int x;  short foo2( ) {  int y; // OK  for( int iValue = 0; iValue < 10; iValue++ ) { // OK  }  return Var3;  }  };  int A::foo1( ) {  int iValue = 6; // OK  return iValue;  } |

## Possible Bugs

|  |  |
| --- | --- |
| 01 | Avoid assigning out-of-range value to char type |
| Description: | Example:  void Foo()  {  char c1 = 0;  c1 = 145; //Violation  // ...  return;  } |
| 02 | Avoid assigning out-of-range value to char type |
| Description: | Example:  void Foo()  {  char uVal = 0;  uVal = -133; //Violation  // ...  return;  } |
| 03 | If a class has pointer members it must define a copy constructor and operator= |
| Description: | Example:  class A { // Violation  int \*a;  };  // with copy constructor  class B { // Violation  int \*b;  public:  B( B& b );  };  // with operator=  class C { // Violation  int \*c;  public:  C& operator=( C& c );  };  Repair:  class A { // OK  int \*a;  public:  A( A& a );  A& operator=( A& a );  };  class B { // OK  int \*b;  public:  B( B& b );  B& operator=( B& b );  };  class C { // OK  int \*c;  public:  C( C& c );  C& operator=( C& c );  }; |
| 04 | Avoid assigning out-of-range value to unsigned char type |
| Description: | Example:  void Foo()  {  unsigned char uVal = 0;  uVal = 289; //Violation  // ...  return;  } |
| 05 | Avoid assigning out-of-range value to unsigned char type. |
| Description: | Example:  void Foo()  {  unsigned char uVal = 0;  uVal = -1; //Violation  // ...  return;  } |
| 06 | Never convert consts to non-consts |
| Description: | Example:  void foo(const int\* x){  int\* y;  y = (int\*)x; // Violation  }  Repair:  void foo(const int\* x){  const int\* y;  y = (const int\*)x; // OK  } |
| 07 | Do not overload functions within a template class |
| Description: | Example:  template <class T> class A  {  public:  int foo( T );  int foo( int ); // Violation  };  Repair:  template <class T> class A  {  public:  int foo( T ); // OK  }; |
| 08 | A pointer to a class may not be converted to a pointer of a second class unless the first class inherits from the second |
| Description: | Example:  class A {};  class B {};  class D : public B {};  void main( ) {  A \*a;  B \*b;  D \*d;  b = (B\*) a; // Violation  d = (D\*) b; // Violation  b = (B\*) d; // OK  }  Repair:  class A {};  class B {};  class D : public B {};  void main( ) {  A \*a;  B \*b;  D \*d;  b = (B\*) d; // OK  } |
| 09 | Do not use the bitwise operator in conditionals expressions |
| Description: | Example:  int Foo (int iVar)  {  if (iVar & 0x01) { //Violation  iVar += 10;  } else {  iVar -= 10;  }  return iVar;  } |
| 10 | Always terminate a case statement with break |
| Description: | Example:  int main()  {  int a = 0;  switch (a) {  case 0:  return 0;  case 1: // Violation  a = 1;  }  }  Repair:  //OK  int main()  {  int a = 0;  switch (a) {  case 0:  return 0;  case 1:  a = 1;  break;  }  } |
| 11 | Do not cast pointers to non-pointers |
| Description: | Example:  void func()  {  int i;  char \* pchar = "hello";  i = (int)pchar; // Violation  } |
| 12 | Do not cast an signed char to an unsigned int. |
| Description: | Example:  bool foo( char ch )  {  if( (unsigned) ch == 0xFF ) { // Violation  return true;  }  return false;  } |
| 13 | Do not compare chars to constants out of char range |
| Description: | Example:  int foobar(char c)  {  if (c > 300) { // Violation  return 3;  }  return 0;  }  Repair:  int foobar(char c)  {  if (c > 25) { // OK  return 2;  }  return 0;  } |
| 14 | Do not compare chars to constants out of char range |
| Description: | Example:  int foobar(char c)  {  if (300 > c) { // Violation  return 3;  }  return 0;  }  Repair:  int foobar(char c)  {  if (25 > c) { // OK  return 2;  }  return 0;  } |
| 15 | Do not call delete on non-pointers |
| Description: | Example:  class Rhino {  public:  Rhino( );  Rhino( char\* );  Rhino( const char\* );  operator char\*( );  };  void func()  {  Rhino r;  delete( r ); // Violation  } |
| 16 | Incorrect End-Of-String (EOS) definition |
| Description: | Example:  #define EOS 0 //Violation  #undef EOS  #define EOS NULL //Violation  #undef EOS  #define EOS '\0' //OK |
| 17 | Don't check floats for equality; check for greater than or less than |
| Description: | Example:  void func(float a, int b)  {  if (a==b) { } // Violation  while (a!=b) { } // Violation  }  Repair:  void func(float a, int b)  {  if (a>=b) { } // OK  while (a<=b) { } // OK  } |
| 18 | Don't check floats for equality; check for greater than or less than |
| Description: | Example:  void func(int a, float b)  {  if (a==b) { } // Violation  while (a!=b) { } // Violation  }  Repair:  void func(int a, float b)  {  if (a>=b) { } // OK  while (a<=b) { } // OK  } |
| 19 | When working with float or double expressions, use less than or equal to or greater than or equal to instead of == |
| Description: | Example:  int main()  {  float a = 1.0;  float b = 2.0;  if (a == b) { // Violation  return 1;  }  return 0;  }  Repair:  int main()  {  float a = 1.0;  float b = 2.0;  if (a <= b) { // OK  return 1;  }  return 0;  } |
| 20 | Don't assign the dividend of two integers to a floating-point type |
| Description: | Example:  void func()  {  int a = 3;  int b = 4;  double d;  d = a / b; // Violation  }  Repair:  void func()  {  int a = 3;  int b = 4;  double d;  d = ((double)a) / b; // OK  } |
| 21 | Avoid assignment in if statement condition |
| Description: | Example:  void foo(int a, int b) {  if ( a = b ) {} // Violation  }  Repair:  void foo(int a, int b) {  if ( a == b ) {} // OK  } |
| 22 | Avoid assigning out-of-range value to char type |
| Description: | Example:  void Foo()  {  char uVal = 157; //Violation  // ...  return;  } |
| 23 | Avoid assigning out-of-range value to unsigned char type |
| Description: | Example:  void Foo()  {  unsigned char uVal = 259; //Violation  // ...  return;  } |
| 24 | A pointer to an abstract class shall not be converted to a pointer of a class that inherits from the abstract class |
| Description: | Example:  class B {  public:  virtual int foo( ) = 0;  };  class D : public B {  public:  int foo( );  };    void main( ) {  B \*b;  D \*d;  d = (D\*) b; // Violation  }  Repair:  class B {  public:  virtual int foo( ) = 0;  };  class D : public B {  public:  int foo( );  };    void main( ) {  B \*b;  D \*d;  } |
| 24 | A pointer to an abstract class shall not be converted to a pointer of a class that inherits from the abstract class |
| Description: | Example:  class B {  public:  virtual int foo( ) = 0;  };  class D : public B {  public:  int foo( );  };    void main( ) {  B \*b;  D \*d;  d = (D\*) b; // Violation  }  Repair:  class B {  public:  virtual int foo( ) = 0;  };  class D : public B {  public:  int foo( );  };    void main( ) {  B \*b;  D \*d;  } |
| 25 | Don't pass possibly null pointers as parameters |
| Description: | Example:  void Foo (int \*ptr1, char \*ptr2)  {  \*ptr1 = 10; // Violation  ptr2 = 0;  return;  }  Repair:  void Foo (float \*ptrF)  {  if( ptrF==0) {  return;  }  \*ptrF = 0; // OK  return;  } |
| 26 | Public member functions shall always return const handles to member data |
| Description: | Example:  class A {  int a;  public:  int\* foo( );  };  int\* A::foo( ) {  return &a; // Violation  }  Repair:  class A {  int a;  public:  const int\* foo( );  };  const int\* A::foo( ) {  return &a; // OK  } |
| 27 | Do not throw from within destructor |
| Description: | Example:  class Foo  {  public:  Foo() {}  ~Foo() { // Violation  throw;  }  }; |
| 28 | If a class has virtual functions it shall have a virtual destructor |
| Description: | Example:  class A { // Violation  public:  ~A( );  virtual int foo( );  };  Repair:  class A { // OK  public:  virtual ~A( );  virtual int foo( );  }; |
| 29 | Whenever a global function is referenced, use the :: operator. |
| Description: | Example:  namespace N  {  void globalFoo1();  }  void globalFoo1()  {  }  void globalFoo2()  {  globalFoo1(); //Violation  N::globalFoo1(); //Violation  }  Repair:  namespace N  {  void globalFoo1();  }  void globalFoo1()  {  }  void globalFoo2()  {  ::globalFoo1(); // OK  ::N::globalFoo1(); // OK  } |
| 30 | Local variables and class variables should have different name. |
| Description: | Example:  class A {  int member;  float var1;  float var2;  short var3;  void foo( ) {  float member; // Violation  }  int foo1( );  short foo2( ) {  for( int var2 = 0; var2 < 10; var2++ ) { // Violation  }  return var3;  }  };  int A::foo1( ) {  int var1 = 6; // Violation  return var1;  }  Repair:  class A {  int member;  float var1;  float var2;  short var3;  void foo( ) {  float local; // OK  }  int foo1( );  short foo2( ) {  for( int index = 0; index < 10; index++ ) { // OK  }  return var3;  }  };  int A::foo1( ) {  int local = 6; // OK  return local;  } |
| 31 | Declared types of formal and actual parameters to functions must match |
| Description: | Example:  void foo(int a, int b)  {  }  void goo(float a, float b)  {  foo(a, b); // Violation  }  Repair:  void foo(int a, int b)  {  }  void goo(int a, int b)  {  foo(a, b); // OK  } |
| 32 | Do not assign function return value to a variable of incompatible type |
| Description: | Example:  float goo();  void foo()  {  int a = 0;  a = goo(); // Violation  }  Repair:  float goo();  void foo()  {  float a = 0.1;  a = goo(); // OK  } |
| 33 | Do not assign incompatible variable types |
| Description: | Example:  void foo()  {  int a = 0;  float b = 1.1;  a = b; // Violation  b = a; // Violation  }  Repair:  void foo()  {  int a = 0;  int b = 1;  a = b; // OK  b = a; // OK  } |
| 34 | Return value of a function must match declared return type |
| Description: | Example:  typedef int int\_t;  typedef float float\_t;  int\_t foo( ) {  float\_t ret = 0;  return ret; // Violation  }  Repair:  typedef int int\_t;  typedef float float\_t;  int\_t foo( ) {  int\_t ret = 0;  return ret; // OK  } |
| 35 | Using mixed types in ternary operator is not allowed |
| Description: | Example:  void foo()  {  int x;  x = ((y > 5) ? 1.1 : 0.2); // Violation  }  Repair:  void foo()  {  int x;  x = ((y > 5) ? 1 : 0); // OK  } |
| 36 | Do not dereference pointer type expressions |
| Description: | Example:  class XX{  public:  int\* sample(){return 0;};  };  class EG{  public:  int\* sample(){return 0;};  };  int \* pFun(int a);  int \* pFun2();  void goo()  {  EG \*p;  EG \*q;  int \*ok;  int \*pt;  int \*pp;  int a=0;  \*pt++; //Violation  a=\*pFun2()\*\*pt; //Violation  a=\*pFun(a\*3); // Violation  \*((q+5)+5)=\*(p+5); //Violation  \*((XX\*)p)->sample(); //Violation  }  Repair:  class XX{  public:  int\* sample(){return 0;};  };  class EG{  public:  int\* sample(){return 0;};  };  int \* pFun(int a);  int \* pFun2();  void goo()  {  EG \*p;  EG \*q;  int \*ok;  int \*pt;  int \*pp;  int a=0;  a=\*pt; //ok  \*pt=a; //ok  a++; //ok  (\*pt+5);//ok  (\*pt+\*pp); //ok  ok = ((XX\*)q)->sample(); //ok  } |
| 37 | The pointer shall not be dereferenced without checking against NULL value |
| Description: | Example:  int\* foo( );  void bar( int\* p ) {  int j;  p = foo( );  j = \*p; // Violation  }  Repair:  int\* foo( );  void bar( int\* p ) {  int j;  p = foo( );  if (p != 0) {  j = \*p; // OK  }  }  References:  Origin: Misra Guidelines - Rule 107 |
| 38 | Do not dereference pointer type expression without checking the expression's value. |
| Description: | Example:  int foosy(int a);  class XX{  public:  int\* sample(){return 0;};  };  class EG{  public:  int\* sample(){return 0;};  };  void moo()  {  EG \*p=0;  foosy(\*((XX\*)p)->sample()); //violation  }  Repair:  int foosy(int a);  class XX{  public:  int\* sample(){return 0;};  };  class EG{  public:  int\* sample(){return 0;};  };  void moo\_ok()  {  EG \*p=0;  if(((XX\*)p)->sample())  {  foosy(\*((XX\*)p)->sample());  }  } |
| 39 | Do not use assignments inside a(b), a[b], cast, and sizeof expressions |
| Description: | Example:  void foo( int i ) {  int\* x;  x[ i = 0 ]; // Violation  foo( i = 0 ); // Violation  i = (int) (x = 0); // Violation  sizeof( i = 0 ); // Violation  }  Repair:  Do not use assignments inside a(b), a[b], cast, sizeof expressions. |
| 40 | Avoid returning handles to function parameters. |
| Description: | Example:  int\* foo1( int x ){  return &x; // Violation  }  int& foo2( int x ){  return x; // Violation  }  Repair:  int foo1( int x ){  return x; // OK  }  int foo2( int x ){  return x; // OK  } |
| 41 | Local variables/parameters and class/parent classes/parent structs variables should have different name. |
| Description: | Example:  class Base {  void foo( ) {  float member; // Violation  }  int foo1( );  short foo2( ) {  for( int var2 = 0; var2 < 10; var2++ ) { // Violation  }  return var3;  }  public:  int member;  float var1;  float var2;  short var3;  };  int Base::foo1( ) {  int var1 = 6; // Violation  return var1;  }  class A : public Base {  void foo( ) {  float member; // Violation  }  int foo1( );  short foo2( ) {  for( int var2 = 0; var2 < 10; var2++ ) { // Violation  }  return var3;  }  };  int A::foo1( ) {  int var1 = 6; // Violation  return var1;  }  Repair:  class Base {  void foo( ) {  float localMember; // OK  }  int foo1( );  short foo2( ) {  for( int iValue = 0; iValue < 10; iValue++ ) { // OK  }  return var3;  }  public:  int member;  float var1;  float var2;  short var3;  };  int Base::foo1( ) {  int iValue = 6; // OK  return iValue;  }  class A : public Base {  void foo( ) {  float localMember; // OK  }  int foo1( );  short foo2( ) {  for( int iValue = 0; iValue < 10; iValue++ ) { // OK  }  return var3;  }  };  int A::foo1( ) {  int iValue = 6; // OK  return iValue;  } |

## Optimization

|  |  |
| --- | --- |
| 07 | Prefer "a @= b" than "a = a @ b", where "@" is +, -, \*, /, % |
| Description: | Example:  void foo( int x, int y ) {  x = x + 2; // Violation  x = x - 2; // Violation  x = x \* 2; // Violation  x = x / 2; // Violation  x = x % 2; // Violation  }  Repair:  void foo( int x, int y ) {  x += 2; // OK  x -= 2; // OK  x \*= 2; // OK  x /= 2; // OK  x %= 2; // OK  } |
| 08 | Prefer "a @= b" than "a = a @ b", where "@" is &, |, ^, <<, >> |
| Description: | Example:  void foo( int x, int y ) {  x = x & 2; // Violation  x = x | 2; // Violation  x = x ^ 2; // Violation  x = x << 2; // Violation  x = x >> 2; // Violation  }  Repair:  void foo( int x, int y ) {  x &= 2; // OK  x |= 2; // OK  x ^= 2; // OK  x <<= 2; // OK  x >>= 2; // OK  } |
| 01 | Do not write unreachable code |
| Description: | Example:  void foo ()  {  int i = 0;  if(1) // Violation  {  i = 1;  } else {  i = 2;  }  return;  } |
| 02 | Do not write unreachable code |
| Description: | Example:  int foo ()  {  int i = 0;  return(i);  i = 7; // Violation  } |
| 03 | Avoid unused local variables |
| Description: | Example:  void func()  {  int i = 0; // Violation  } |
| 04 | Eliminate unused parameters |
| Description: | Example:  int Foo(int i, int k) { // Violation  i = 5;  return i;  }  Repair:  int Bar(int i, int j) { // OK  i += j;  return i;  } |
| 05 | Eliminate unused private member variables |
| Description: | Example:  class Foo {  public:  Foo(): x(0), y(0) { }  int getX() {  return x;  }  private:  int x;  int y; // Violation  }; |
| 05 | Eliminate unused private member variables |
| Description: | Example:  class Foo {  public:  Foo(): x(0), y(0) { }  int getX() {  return x;  }  private:  int x;  int y;  int z; // Violation  }; |
| 06 | Avoid unused local variables or variables which have no side effect |
| Description: | Example:  void func() {  int i = 0; // Violation  }  int i = 0;  class A {  public:  A(){};  };  class B {  public:  B(){i++;};  };  void goo() {  A a; // Violation  B b; // OK  }  Repair:  void func() { // OK  }  int i = 0;  class A {  public:  A(){};  };  class B {  public:  B(){i++;};  };  void goo() { // OK  B b;  } |
| 09 | Do not define class/struct/union inside function implementation |
| Description: | Example:  void foo(){  class A { // Violation  // ...  };  // ...  }  Repair:  class A { // OK  // ...  };  void foo(){  // ...  } |
| 10 | Do not declare variables in "if", "for", "while", and "do while" statement |
| Description: | Example:  int bar( int& x );  int lookup( int& x );  void foo( int x ){  if (x == 0) {  int y; // Violation  y = bar(x);  }  for (x = 0; x < 2;x++) {  int y; // Violation  y = lookup(x);  bar(y);  }  while (x < 0) {  int y; // Violation  y = bar(x);  }  do {  int y; // Violation  y = bar(x);  } while (x < 0);  }  Repair:  int bar( int& x );  int lookup( int& x );  void foo( int x ){  int y; // OK  if (x == 0) {  y = bar(x);  }  for (x = 0; x < 2;x++) {  y = lookup(x);  bar(y);  }  while (x < 0) {  y = bar(x);  }  do {  y = bar(x);  } while (x < 0);  } |

## Portability

|  |  |
| --- | --- |
| 02 | Assigning result of ints operation to long without casting at least one of the ints is not allowed |
| Description: | Example:  void foo() {  int a = 72;  int b = 55;  long result = 0;  result = (long)(a\*b); // Violation  }  Repair:  void foo() {  int a = 72;  int b = 55;  long result = 0;  result = (long)((long)a\*b); //OK  } |
| 03 | Do not cast pointer to UINT type |
| Description: | Example:  typedef unsigned int UINT;  UINT foo(){  char \*buf;  return (UINT)buf; //Violation  }  Repair:  Do not cast a pointer to a UINT type. |
| 04 | Don't use hard coded value of offset in structures |
| Description: | Example:  #include <stddef.h>  #define ARCH32 true  #ifdef ARCH32  #define \_\_int3264 int //32-bit type on win32 and unix32  #else //ARCH64  #ifdef win64  #define \_\_int3264 \_\_int64 //64-bit type on win64 (warning: long on win64 is 32-bit type)  #else //unix64  #define \_\_int3264 long //64-bit type on unix64 (long on unix64 is 64-bit type)  #endif  #endif  struct myStruct {  void \*ptr;  int i;  };  void foo() {  myStruct s;  myStruct \*ps = new myStruct();  int j;  s.i = 22;  ps->i = 22;  j= \*(int \*)((unsigned \_\_int3264)&s + sizeof(void\*)); //Violation  j= \*(int \*)((unsigned \_\_int3264)&s + 4 ); //Violation  j= \*(int \*)((unsigned \_\_int3264)&s.i + 0 ); //Violation  j= \*(int \*)((unsigned \_\_int3264)ps + sizeof(void\*)); //Violation  j= \*(int \*)((unsigned \_\_int3264)ps + 4); //Violation  j= \*(int \*)((unsigned \_\_int3264)&ps->i + 0); //Violation  }  Repair:  #include <stddef.h>  #define ARCH32 true  #ifdef ARCH32  #define \_\_int3264 int //32-bit type on win32 and unix32  #else //ARCH64  #ifdef win64  #define \_\_int3264 \_\_int64 //64-bit type on win64 (warning: long on win64 is 32-bit type)  #else //unix64  #define \_\_int3264 long //64-bit type on unix64 (long on unix64 is 64-bit type)  #endif  #endif  struct myStruct {  void \*ptr;  int i;  };  void foo() {  myStruct s;  myStruct \*ps = new myStruct();  int j;  s.i = 22;  ps->i = 22;  j= \*(int \*)((unsigned \_\_int3264)&s + offsetof(myStruct,i)); //OK  j= \*(int \*)((unsigned \_\_int3264)ps + offsetof(myStruct,i)); //OK  } |
| 05 | Don't use hex constants |
| Description: | Example:  #ifdef \_WIN64  #define CONST3264(a) (a##i64)  #else // unix64/32 & win32  #define CONST3264(a) (a##L)  #endif  void foo() {  long Val1 = 0xFFFFFFFF; //Violation  }  Repair:  #ifdef \_WIN64  #define CONST3264(a) (a##i64)  #else // unix64/32 & win32  #define CONST3264(a) (a##L)  #endif  void foo() {  long Val2 = CONST3264(-1); //OK  } |
| 06 | Do not use implicit truncation |
| Description: | Example:  void foo(){  int iVal = 72;  long lVal = 6;  iVal = lVal; //Violation  iVal = (int)lVal; //Violation  }  Repair:  Do not assign a long int to an int. |
| 07 | Constant assignment to long should not involve int literals |
| Description: | Example:  void foo()  {  long a;  a &= ~7; // Violation  }  Repair:  void foo()  {  long a;  a &= ~7L; // OK  } |
| 07 | Constant assignment to long should not involve int literals. |
| Description: | Example:  void foo()  {  long a;  a &= 7; // Violation  }  Repair:  void foo()  {  long a;  a &= 7L; // OK  } |
| 08 | Do not assign literal to type long |
| Description: | Example:  void foo()  {  long lVal = 1; //Violation  }  Repair:  void foo()  {  long lVal = 1L; //OK  } |
| 09 | Do not cast pointer to int type |
| Description: | Example:  void foo()  {  char \*p;  int g=(int)p; //Violation  }  Repair:  Do not cast pointer to int type. |
| 10 | Incompatible cast |
| Description: | Example:  void f2005() {  int \*intPointer;  long \*longPointer;  longPointer = (long \*)intPointer; // Violation  } |
| 11 | Do not assign a long value to a double |
| Description: | Example:  void foo() {  double doubleValue;  long longValue = 1L;  doubleValue = longValue; // Violation  }  Repair:  Do not assign a long value to a double. |
| 12 | Do not cast a long value to a double |
| Description: | Example:  void foo()  {  long longValue = 1L;  (double)longValue; // Violation  }  Repair:  Do not cast a long value to a double. |
| 12 | Do not cast a long value to a double |
| Description: | Example:  void moo (double) {}  void foo () {  long longValue = 1L;  moo(longValue); // Violation  }  Repair:  Do not cast a long value to a double. |
| 13 | Incompatible cast |
| Description: | Example:  void foo() {  int \*intPointer;  long \*longPointer;  intPointer = (int \*)longPointer; // Violation  }  Repair:  A long pointer should not be passed to a int pointer. |
| 14 | Don't use i64 or L suffixes directly |
| Description: | Example:  void goo(double a);  double foo1()  {  double bal = 5l; // Violation  goo(4567L); // Violation  return 567l; // Violation  }  #define V1 0x000000000000000cL //Violation  double a1 = 456l; // Violation  double r = 45i64; // Violation  Repair:  #ifdef \_WIN64  #define CONST3264(a) (a##i64)  #else  #define CONST3264(a) (a##L)  #endif  void goo(double a);  double foo1()  {  double bal = CONST3264(5); // OK  goo(CONST3264(4567)); // OK  return CONST3264(567); // OK  }  #define V1 CONST3264(0x000000000000000c) //OK  double a1 = CONST3264(456); // OK  double r = CONST3264(45); // OK |
| 14 | Don't use i64 or L suffixes directly |
| Description: | Example:  void goo(double a);  double foo1()  {  double bal = 5l; // Violation  goo(4567L); // Violation  return 567l; // Violation  }  #define V1 0x000000000000000cL //Violation  double a1 = 456l; // Violation  double r = 45i64; // Violation  Repair:  #ifdef \_WIN64  #define CONST3264(a) (a##i64)  #else  #define CONST3264(a) (a##L)  #endif  void goo(double a);  double foo1()  {  double bal = CONST3264(5); // OK  goo(CONST3264(4567)); // OK  return CONST3264(567); // OK  }  #define V1 CONST3264(0x000000000000000c) //OK  double a1 = CONST3264(456); // OK  double r = CONST3264(45); // OK |
| 15 | Use of explicit cast on operand of type long |
| Description: | Example:  void foo() {  int int1, int2, int3;  long longValue;  int2 = ((int)longValue)/int1; // Violation: longValue is truncated to 32 bits before divide  int3 = (int)(longValue/int1); // OK: the result of the division is truncated to 32 bits.  }  Repair:  Do not use of explicit cast on operand of type long. |
| 16 | Avoid using union |
| Description: | Example:  union myUnion { //Violation  long l;  char bytes[4];  };  Repair:  Don't use union. |
| 17 | Avoid using long |
| Description: | Example:  #define ARCH32 true  #ifndef \_\_int3264  #ifdef ARCH32  #define \_\_int3264 int //32-bit type on win32 and unix32  #else //ARCH64  #ifdef win64  #define \_\_int3264 \_\_int64 //64-bit type on win64  #else //unix64  #define \_\_int3264 int64\_t //64-bit type on unix64  #endif  #endif  #endif  long lVal; //Violation  Repair:  #define ARCH32 true  #ifndef \_\_int3264  #ifdef ARCH32  #define \_\_int3264 int //32-bit type on win32 and unix32  #else //ARCH64  #ifdef win64  #define \_\_int3264 \_\_int64 //64-bit type on win64  #else //unix64  #define \_\_int3264 int64\_t //64-bit type on unix64  #endif  #endif  #endif  \_\_int3264 newlVal; //OK |
| 18 | Do not use a #define that prevents the compiler from checking types |
| Description: | Example:  #define V1 20 //Violation  Repair:  const int V2 = 20; //OK |
| 18 | Do not use a #define that prevents the compiler from checking types |
| Description: | Example:  #define V 20000L // Violation  void foo() {  int i = V;  }  Repair:  Do not use a #define that prevents the compiler from checking types. |
| 18 | Do not use a #define that prevents the compiler from checking types |
| Description: | Example:  #define MAX\_COMPUTERNAME\_LENGTH 10  #define MACHINE\_NAME\_LEN MAX\_COMPUTERNAME\_LENGTH+2 // Violation  Repair:  Do not use a #define that prevents the compiler from checking types. |
| 18 | Do not use a #define that prevents the compiler from checking types |
| Description: | Example:  #define MAX\_COMPUTERNAME\_LENGTH 10  #define MACHINE\_NAME\_LEN 2+MAX\_COMPUTERNAME\_LENGTH // Violation  Repair:  Do not use a #define that prevents the compiler from checking types. |
| 26 | Do not use implicit truncation |
| Description: | Example:  void foo(int iVal){  long lVal ;  //.. set big value for lVal  foo(lVal); // Violation  foo((int)lVal); // Violation  }  Repair:  void foo(int iVal)  {  int intValue;  foo(intValue); //OK  } |

## Initialization

|  |  |
| --- | --- |
| 01 | Headers should not contain any initialization. |
| Description: | Example:  // Header file  int i = 0; // Violation  int j; // OK |
| 02 | Do not initialize unsigned integer variables with signed constants. |
| Description: | Example:  void foo()  {  unsigned int x = 21; // Violation  unsigned int y = -21; // Violation  }  Repair:  void foo()  {  unsigned int z = 21u; // OK  } |
| 03 | Initialize all variables. |
| Description: | Example:  int main()  {  int a; // Violation  int b = 0; // OK  }  Repair:  int main()  {  int a = 0; // OK  int b = 0; // OK  } |
| 04 | Initialize all pointer variables. |
| Description: | Example:  void foo() {  int \*i; // Violation  }  Repair:  void foo() {  int \*i = 0; // OK  } |
| 05 | Do not initialize a reference to an object whose address can be changed. |
| Description: | Example:  void foo()  {  int \*ptr = 0;  int &rptr = \*ptr; // Violation  } |
| 06 | All member variables should be initialized in constructor. |
| Description: | Example:  class C{    C();  int a;  int b;  float c;  };  C::C():a(0) //Violation c member variable is not initialized in constructor  {  b=1;  }  Repair:  class C{ //Ok    C();  int a;  int b;  float c;  };  C::C():a(0),c(1.0)  {  b=1;  } |
| 07 | Initialization of class type variables |
| Description: | Example:  class B //No constructor  {  public:  static int s;  int b;  int a;  };  int main()  {  B bb; //VIOLATION  return 0;  }  Repair:  class A  {  public:  A(){}  private:  static int w;  int b;  int a;  };  class C //No fields  {  public:  int foo();  void qwe(int x);  };  class D //Only static  {  public:  static int s;  static int b;  static int a;  };  int main()  {  C cc; //untested, no members  A a; //OK  D dd; //untested, only static  return 0;  } |
| 08 | Initialization of dynamically allocated class objects |
| Description: | Example:  class B //No constructor  {  public:  static int s;  int b;  int a;  };  B\* foo(B\* b) {  B\* a=new B(); //VIOLATION  return new B(); //VIOLATION  }  int main()  {  foo(new B()); //VIOLATION  return 0;  }  Repair:  class A  {  public:  A(){}  static int w;  int b;  int a;  };  A\* foo2() {  A\* a=new A(); //OK  return new A(); //OK  } |

## Security

|  |  |
| --- | --- |
| 29 | All exceptions should be rethrown or logged with standard logger |
| Description: | Example:  #include <stdlib.h>  class LoginException{  public:  int tmp;  };  class CLogin {  public:  void login(){};  };  void log( char \*text){};  void foo( int i ) {  CLogin lc;  try {  lc.login();  } catch (LoginException le1) { // Violation  // login failed  le1.tmp = 0;  exit( 1 );  }  }  Repair:  #include <stdlib.h>  class LoginException{  public:  int tmp;  };  class CLogin {  public:  void login(){};  };  void log( char \*text){};  void foo( int i ) {  CLogin lc;  try {  lc.login();  } catch (LoginException le1) { // OK  // login failed  log( "login failed" );  // or throw new LoginException();  le1.tmp = 0;  exit( 1 );  }  } |
| 30 | Avoid passing user input into methods as parameters |
| Description: | Example:  #include <iostream>  using namespace std;  #include <stdio.h>  void function( int param );  void foo( ) {  int var1 = 0;  int var2 = 0;  cin >> var1;  function( var1 ); // Violation  scanf( "%d", &var2 );  function( var2 ); // Violation  }  Repair:  #include <iostream>  using namespace std;  #include <stdio.h>  void function( int param );  void foo( ) {  int var1 = 0;  int var2 = 0;  cin >> var1;  // function( var1 ); // OK  // part of code manipulating var1 without using function with var1 as parameter  scanf( "%d", &var2 );  // function( var2 ); // OK  // part of code manipulating var2 without using function with var2 as parameter  } |
| 19 | Avoid passing output into exception messages |
| Description: | Example:  #include <stdlib.h>  #include <stdio.h>  class LoginException{  public:  int tmp;  char\* getMessage();  };  class CLogin {  public:  void login(){};  };  void log( char \*text){};  void foo( int i ) {  CLogin lc;  try {  lc.login();  } catch (LoginException le1) {  // login failed  ::printf( le1.getMessage() ); // Violation  exit( 1 );  }  }  Repair:  #include <stdlib.h>  #include <stdio.h>  class LoginException{  public:  int tmp;  char\* getMessage();  };  class CLogin {  public:  void login(){};  };  void log( char \*text){};  void foo( int i ) {  CLogin lc;  try {  lc.login();  } catch (LoginException le2) { // OK  exit( 1 );  }  } |
| 08 | Avoid "protected" data members |
| Description: | Example:  class A  {  protected:  int iProtectedData; // Violation  };  Repair:  class A  {  private:  int iPrivateData;  protected:  const int accessData();  }; |
| 09 | Avoid "public" data members that could be accessed by untrusted classes |
| Description: | Example:  class A  {  public:  int iPublicData; // Violation  };  Repair:  class A  {  private:  int iPrivateData;  const int accessData();  }; |
| 13 | Avoid using shell functions in your program |
| Description: | Example:  #include <process.h>  void main( void )  {  \_execl( "c:\\command.com", "", "\_execl", "", "" ); // Violation  }  Repair:  Do not use unsecured system functions in your program.  References:  http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/handle-metacharacters.html |
| 19 | Avoid passing output into exception messages |
| Description: | Example:  #include <stdlib.h>  #include <stdio.h>  #include <iostream>  using namespace std;  class LoginException{  public:  int tmp;  char\* getMessage() { return "error"; };  };  class CLogin {  public:  void login(){};  };  void log( char \*text){};  void foo( int i ) {  CLogin lc;  char\* msg = "Bad login";  try {  lc.login();  } catch (LoginException le1) {  // login failed  printf( le1.getMessage() ); // Violation  cout << msg; // Violation  exit( 1 );  }  }  int main() { return 0; }  Repair:  #include <stdlib.h>  #include <stdio.h>  #include <iostream>  using namespace std;  class LoginException{  public:  int tmp;  char\* getMessage() { return "error"; };  };  class CLogin {  public:  void login(){};  };  void log( char \*text){};  void foo( int i ) {  CLogin lc;  char\* msg = "Bad login";  try {  lc.login();  } catch (LoginException le2) { // OK  exit( 1 );  }  }  int main() { return 0; } |
| 30 | Avoid passing user input into methods as parameters |
| Description: | Example:  #include <iostream>  #include <stdio.h>  using namespace std;  void function( int param );  void foo( int param ) {  int var1;  int var2=0;  cin >> var1;  function( var1 ); // Violation  scanf( "%d", &var2 );  function( var2 ); // Violation  }  Repair:  #include <iostream>  #include <stdio.h>  using namespace std;  void function( int param );  void foo( ) {  int var1 = 0;  int var2 = 0;  cin >> var1;  // part of code manipulating var1 without using function with var1 as parameter  scanf( "%d", &var2 );  // part of code manipulating var2 without using function with var2 as parameter  } |
| 31 | Do not use 'syslog' function for logging purposes |
| Description: | Example:  #include <syslog.h>  int main()  {  syslog(LOG\_INFO, "Some info"); // Violation  }  Repair:  Do not use 'syslog' function in your code. |
| 32 | Do not use mbstowcs() function |
| Description: | Example:  #include <stdlib.h>  #include <stdio.h>  void main( void )  {  int i;  char \*pmbhello = (char \*)malloc( MB\_CUR\_MAX );  wchar\_t \*pwc = (wchar\_t \*)malloc( sizeof( wchar\_t ));    i = mbstowcs( pwc, pmbhello, MB\_CUR\_MAX ); // Violation  }  Repair:  Do not use mbstowcs(). |
| 20 | Avoid passing dynamically created strings into exec |
| Description: | Example:  #include <stdio.h>  #include <process.h>  void foo()  {  char \*param = "C:\\command.com";  \_execl( param, "", "\_execl", "", NULL ); // Violation  }  Repair:  #include <stdio.h>  #include <process.h>  void foo()  {  \_execl( "C:\\command.com", "", "\_execl", "", NULL ); // OK  } |
| 21 | Avoid passing dynamically created strings into exec |
| Description: | Example:  #include <stdio.h>  #include <process.h>  #include <string.h>  class cll {  public:  char\* text();  };  void foo( char\* prog ){  cll \*v1 = 0;  char \* v4=v1->text();  prog = strcat( "c:\\", "command.com" );  \_execl( prog, "", "\_execl", "", NULL ); // Violation  \_execl( v1->text(), "", "\_execl", "", NULL );// Violation  \_execl( v4, "", "\_execl", "", NULL ); // Violation  }  Repair:  #include <stdio.h>  #include <process.h>  void foo( char\* prog )  {  \_execl( "C:\\command.com", "", "\_execl", "", NULL ); // OK  } |
| 33 | GetTempPath() may return the current directory or the windows directory |
| Description: | Example:  #include <afx.h>  const char \* virus\_temp\_sig = "XXX";  char \* get\_temp\_file()  {  char wintemp\_path[1024];  char \*temp\_path=new char[1024];    GetTempPath(1024,wintemp\_path); // Violation  GetTempFileName(wintemp\_path,virus\_temp\_sig,1234,temp\_path);  return temp\_path;  }  Repair:  Do not use the GetTempPath(),GetTempPathA(), GetTempPathW() functions,  or at least be careful what you place in locations returned by them. |
| 04 | Avoid loading Dynamic Link Libraries |
| Description: | Example:  #include <windows.h>  void main( void )  {  LoadLibrary("mydll.dll"); // Violation  }  Repair:  Do not load Dynamin Link Libraries in source code. |
| 03 | Usage of system properties should be restricted |
| Description: | Example:  #include <stdlib.h>  void main( void )  {  char \*envvar;  envvar = getenv( "LIB" ); // Violation  \_putenv( "LIB=c:\\mylib;c:\\yourlib" ); // Violation  }  Repair:  Do not use system properties and environment variables in source code. |
| 05 | Avoid using data in file outside namespaces |
| Description: | Example:  int var = 0; // Violation  void globalfoo() // Violation  {  }  class A // Violation  {  int i;  void foo();  };  Repair:  namespace name1  {  int var = 0; // OK  void globalfoo() // OK  {  }  class A // OK  {  int i;  void foo();  };  } |
| 02 | Avoid functions which use random numbers from standard C library |
| Description: | Example:  #include <stdlib.h>  #include <time.h>  void main( void )  {  srand((unsigned)time( NULL )); // Violation  int y = rand(); // Violation  }  Repair:  Remove usage of rand(), random(), randomize(), srand() functions. |
| 06 | Avoid using functions printf/wprintf with only one variable parameter |
| Description: | Example:  #include <stdio.h>  void foo( )  {  char \*cap = {"caption"};  unsigned short d=8;  printf(cap); // Violation  wprintf(&d); // Violation  }  Repair:  #include <stdio.h>  void foo( )  {  char \*cap = {"caption"};  unsigned short d=8;  int k=7;  printf("abc:%d",k); // OK  printf("%s\n”, cap); // OK  wprintf(L"abc:%d",k);// OK  wprintf(L"abcd"); // OK  } |
| 01 | Avoid functions which use time from standard C library |
| Description: | Example:  #include <time.h>  void dangerFunction(time\_t time);  int main ()  {  time\_t rawtime;  tm \* ptm;  time ( &rawtime ); // Violation  ptm = gmtime ( &rawtime ); // Violation  dangerFunction(rawtime);  return 0;  }  Repair:  Remove usage of time-related functions from the standard C library |
| 07 | Avoid functions which use time from MFC library |
| Description: | Example:  #include <afx.h>  void dangerFunction(time\_t t);  int main () {  CTime t(NULL);  time\_t Time = t.GetTime(); // Violation  t.GetHour(); // Violation  t.GetMinute(); // Violation  dangerFunction(Time);  return 0;  } |
| 10 | Avoid using functions fprintf/fwprintf with only two parameters, when second parameter is a variable |
| Description: | Example:  #include <stdio.h>  void foo()  {  FILE \*stream;  char c = '\n';  unsigned short d=8;  stream = fopen( "fprintf.out", "w" );  fprintf( stream,&c); // Violation  fwprintf(stream,&d); // Violation  fclose( stream );  }  Repair:  #include <stdio.h>  void foo()  {  FILE \*stream;  int i = 10;  char s[] = "example string";  char c = '\n';  unsigned short d=8;  stream = fopen( "fprintf.out", "w" );  fprintf( stream,"abcd"); // OK  fprintf( stream, "%s%c", s, c );// OK  fprintf( stream, "%d\n", i ); // OK    fwprintf(stream,L"abc:%d",i); // OK  fwprintf(stream,L"abcd"); // OK    fclose( stream );  } |
| 11 | Avoid using data() function from 'string' class of standard library |
| Description: | Example:  #include <string>  void bar(const char\* param);    void foo()  {  using namespace std;  string str1 ( "example string" );  bar(str1.data()); // Violation    string \*ptr;  ptr=&str1;  const char \*ptr2 = ptr->data(); // Violation  }  Repair:  #include <string>  void bar(const char\* param);    void foo()  {  using namespace std;  string str1 ( "example string" );  bar(str1.c\_str()); // OK    string \*ptr;  ptr=&str1;  const char \*ptr2 = ptr->c\_str(); // OK  }  References:  http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/library-c.html#STD-STRING |
| 12 | Avoid using vfork() function |
| Description: | Example:  #include <unistd.h>  void main( void )  {  vfork( ); // Violation  }  Repair:  #include <unistd.h>  void main( void )  {  fork( ); // OK  }  References:  http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/avoid-vfork.html |
| 14 | Avoid using unsafe string functions |
| Description: | Example:  #include <string.h>  void main( void )  {  char\* str1 = "testcase";  char\* str2 = "testcase";  char\* str3=0;  str3 = strcat( str1, str2 ); // Violation  }  Repair:  #include <string.h>  void main( void )  {  char\* str1 = "testcase";  char\* str2 = "testcase";  char\* str3=0;  str3 = strncat( str1, str2, 16 ); // OK  }  References:  http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/dangers-c.html |
| 15 | Avoid using unsafe string functions |
| Description: | Example:  #include <stdio.h>  void main( void )  {  char\* str = 0;  scanf( "%s", str ); // Violation  }  Repair:  Remove usage of unsafe string functions.  References:  http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/dangers-c.html |
| 16 | Avoid use scanf and fscanf functions without describing size of variables in format |
| Description: | Example:  #include <stdio.h>  void main()  {  char tcChar[10];  char cChar;  float fFloat;  FILE \*stream;  stream = fopen( "fprintf.out", "w" );    scanf("%s",&tcChar); // Violation  scanf("%f",&fFloat); // Violation  fscanf(stream,"%s",&tcChar); // Violation  fscanf(stream,"%c%f",&cChar,&fFloat); // Violation  }  Repair:  #include <stdio.h>  void main()  {  char tcChar[10];  char cChar;  float fFloat;  FILE \*stream;  stream = fopen( "fprintf.out", "w" );  scanf("%c",&cChar); // OK  scanf("%10s",&tcChar); // OK  scanf("%5f",&fFloat); // OK    fscanf(stream,"%c",&cChar); // OK  fscanf(stream,"%10s",&tcChar); // OK  fscanf(stream,"%5f",&fFloat); // OK  } |
| 34 | Avoid using InitializeCriticalSection |
| Description: | Example:  #define \_WIN32\_WINNT 0x0501  #include <afx.h>  class Queue {  public:  Queue(int limit)  {  InitializeCriticalSection(&lock); // Violation  }    ~Queue()  {  ::DeleteCriticalSection(&lock);  }    void AddTail()  {  ::EnterCriticalSection(&lock);  ::LeaveCriticalSection(&lock);  }  protected:  CRITICAL\_SECTION lock;  };  Repair:  #define \_WIN32\_WINNT 0x0501  #include <afx.h>  class Queue1 {  public:  Queue1(int limit)  {  if (!InitializeCriticalSectionAndSpinCount(&lock, 0x80000400) ) // OK  {  exit(0);  }  }    ~Queue1()  {  ::DeleteCriticalSection(&lock);  }    void AddTail()  {  ::EnterCriticalSection(&lock);  ::LeaveCriticalSection(&lock);  }  protected:  CRITICAL\_SECTION lock;  }; |
| 35 | Avoid using unsafe thread methods |
| Description: | Example:  #include <unistd.h>  void foo()  {  char \*name = ttyname(1); // Violation  }  Repair:  #include <unistd.h>  void foo1()  {  char name[100];  int status = ttyname\_r(1,name,100); // OK  } |
| 36 | Do not use 'setuid' in source code |
| Description: | Example:  #include <sys/types.h>  #include <unistd.h>  int main()  {  setuid(34); // Violation  }  Repair:  #include <sys/types.h>  #include <unistd.h>  int main()  {  seteuid(34); // OK  } |
| 37 | Don't use chmod(), chown(), chgrp() |
| Description: | Example:  #include <sys/stat.h>  #include <sys/types.h>  void foo(const char\* path)  {  chmod(path, S\_IRWXU); // Violation  }  Repair:  #include <sys/stat.h>  #include <sys/types.h>  void goo(int fildes)  {  fchmod(fildes, S\_IRWXU); // OK  } |
| 38 | Standard random number generators should not be used to generate randomness for security reasons |
| Description: | Example:  #include <stdlib.h>  void goo()  {  double rnumber = drand48(); // Violation  }  Repair:  Do not use standard random number generators. |
| 39 | Do not use obsolete C routine ulimit() |
| Description: | Example:  #include <ulimit.h>  void foo(int cmd, long newlimit)  {  ulimit(cmd, newlimit); // Violation  }  Repair:  #include <sys/types.h>  #include <sys/resource.h>  #include <unistd.h>  void foo(int who, struct rlimit \*rlim)  {  getrlimit(who, rlim); // OK  } |
| 40 | Avoid using 'getpw' function in program code |
| Description: | Example:  #include <pwd.h>  #include <sys/types.h>  int main()  {  char\* buf;  getpw(0,buf); // Violation  return 0;  }  Repair:  #include <pwd.h>  #include <sys/types.h>  int main()  {  char\* buf;  struct passwd \* ps = getpwuid(0); // OK  return 0;  } |
| 41 | Do not use 'cuserid' function |
| Description: | Example:  #include <unistd.h>  int main()  {  char \* w = cuserid("userName"); // Violation  }  Repair:  #include <sys/types.h>  #include <pwd.h>  int main()  {  struct passwd \* t = getpwuid(12); // OK  } |
| 42 | Avoid using obsolete C routine 'usleep' |
| Description: | Example:  #include <unistd.h>  int main()  {  usleep(7); // Violation  return 0;  }  Repair:  #include <time.h>  void foo(timespec \*req, timespec \*rem)  {  nanosleep(req, rem); // OK  } |
| 43 | Usage of functions which do not properly handle non-NULL terminated strings is not allowed |
| Description: | Example:  #include <mbstring.h>  #include <tchar.h>  void goo()  {  char buffer[61] = "string example";  int len;  len = strlen( buffer ); // Violation  len = \_tcslen( buffer ); // Violation  len = \_mbslen(reinterpret\_cast<const unsigned char\*>(buffer)); // Violation  len = wcslen( reinterpret\_cast<const unsigned short\*>(buffer)); // Violation  }  Repair:  Do not use strlen, \_tcslen, \_mbslen, and wcslen functions. |
| 44 | Avoid using environment variables |
| Description: | Example:  #include <locale.h>  void foo()  {  setlocale(LC\_ALL,""); // Violation  }  Repair:  Do not use described environment variables. |
| 45 | Don't trust any value of command line if attacker can set them |
| Description: | Example:  int main(int argc, char \*\*argv)  {  int local = 0;  local += argc; // Violation  return 1;  }  Repair:  int main() // OK  {  } |
| 46 | Never use unfiltered data from an untrusted user as the format parameter |
| Description: | Example:  void foo()  {  printf(stringFromUntrustedUser); // Violation  }  Repair:  void foo()  {  printf(„%s”,stringFromUntrustedUser); // OK  } |
| 47 | Usage of race condition functions is not allowed |
| Description: | Example:  #include <unistd.h>  int main() {  link("oldpath", "newpath"); // Violation  unlink("/pathname"); // Violation  mktemp("template");  return 0;  }  Repair:  Remove usage of race condition functions. |

## Misra 2004(137)

|  |  |
| --- | --- |
| 2004-2\_1 | Assembly language shall be encapsulated and isolated |
| Description: | Example:  /\* compilable by cl.exe \*/  void asmCall2() /\* Violation \*/  {  int g = 0;  \_\_asm{  mov eax, 01h  int 10h  }  }  Repair:  void asmCall() /\* OK \*/  {  \_\_asm{  mov eax, 01h  int 10h  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 2 |
| 2004-2\_2 | Source code shall only use /\* ... \*/ style comments |
| Description: | Example:  #define comment // Violation  // Violation  Repair:  /\* OK \*/  /\*  \* OK  \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 2 |
| 2004-2\_3 | The character sequence /\* shall not be used within a comment |
| Description: | Example:  /\* Violation - comment shall not be nested violation here /\* \*/  Repair:  /\* OK - C-style comment \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 2 |
| 2004-2\_4 | Sections of code should not be "commented out" |
| Description: | Example:  void foo() {  int dVal = 5;  /\* Violation \*/  /\*  if (dVal==0) {  /\* ... \*/  }  \*/  }  Repair:  Do not “comment out?sections of code.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 2 |
| 2004-3\_4 | All uses of the #pragma directive shall be documented and explained |
| Description: | Example:  #pragma /\* Violation \*/  /\* comment in wrong place \*/  #pragma TEST /\* Violation \*/  /\* comment in wrong place  \*/  #pragma TEST /\* Violation \*/  Repair:  /\* OK \*/  #pragma TEST  /\* OK - multiline comment  \*/  #pragma TEST  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 3 |
| 2004-3\_5 | Do not mix bit-fields other data within the same structure |
| Description: | Example:  struct message { /\* Violation \*/  signed int little: 4;  unsigned int x\_set: 1;  int size;  };  Repair:  struct message { /\* OK \*/  signed int little: 4;  unsigned int x\_set: 1;  };  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 3 |
| 2004-4\_1 | Only those escape sequences that are defined in the ISO C standard shall be used |
| Description: | Example:  #include <stdio.h>  main()  {  printf("ABCD\u"); // Violation  printf("abcd\hgf"); // Violation  printf("\k"); // Violation  }  Repair:  Do not use escape sequences not defined in the ISO C standard.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 4 |
| 2004-4\_2 | Trigraphs shall not be used |
| Description: | Example:  ??=define TEST 1 /\* Violation \*/  void foo() {  const char \* s = "(Date should be in the form ??-??-??)"; /\* Violation \*/  }  Repair:  #define TEST 1 /\* OK \*/  void foo() {  const char \* s = "(Date should be in the form ~~]"; /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 4 |
| 2004-5\_1 | Identifiers (internal and external) shall not rely on the significance of more than 31 characters |
| Description: | Example:  void this\_is\_not\_ok\_11111111111111111(); /\* Violation \*/  struct ThisClassHasANameThatIsTooBig11111 /\* Violation \*/  {  int a;  };  Repair:  void foo(); /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 5 |
| 2004-5\_2 | Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier |
| Description: | Example:  int x;  void foo()  {  int x; /\* Violation \*/  x = 3;  }  Repair:  Avoid hiding names of global variables.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 5 |
| 2004-5\_2 | Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier |
| Description: | Example:  int foo()  {  int a;  {  int a; /\* Violation \*/  }  }  Repair:  Avoid hiding names of local variables.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 5 |
| 2004-5\_3 | Do not reuse typedef names |
| Description: | Example:  typedef int MyInt;  void foo()  {  double MyInt; /\* Violation \*/  }  Repair:  typedef int MyInt;  void foo()  {  double MyVar; /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 5 |
| 2004-5\_3 | Do not reuse typedef names as a typedef name |
| Description: | Example:  typedef unsigned char uint8\_t;  void foo()  {  typedef unsigned char uint8\_t; /\* Violation \*/  }  Repair:  Do not use the same names for different typedefs.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 5 |
| 2004-5\_4 | A tag name shall be a unique identifier |
| Description: | Example:  struct stag { int a; };  void stag(void) /\* Violation \*/  {  int stag ; /\* Violation \*/  }  Repair:  Do not use tag names for any other purpose within the program.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 5 |
| 2004-5\_4 | A tag name shall be a unique identifier |
| Description: | Example:  struct stag { int a; int b; };  void foo1(){  union stag { int a; float b; }; /\* Violation \*/  }  void foo2(void)  {  struct stag { int a; }; /\* Violation \*/  }  Repair:  Do not use the same names for different tags.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 5 |
| 2004-5\_5 | No object or function identifier with static storage duration should be reused |
| Description: | Example:  static float a;  static void foo()  {  int a; /\* Violation \*/  int foo; /\* Violation \*/  }  Repair:  Do not use static identifiers for any other purpose within the program.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 5 |
| 2004-5\_5 | No object or function identifier with static storage duration should be reused |
| Description: | Example:  static float a;  static void foo()  {  static int a ; /\* Violation \*/  static int foo; /\* Violation \*/  }  Repair:  Do not use the same names for different static object or functions.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 5 |
| 2004-5\_6 | No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure and union member names |
| Description: | Example:  typedef struct MyStruct /\* tags name space \*/  {  int Member; /\* members name space \*/  } MyStruct; /\* Violation - identifiers name space\*/  Repair:  typedef struct MyStruct /\* tags name space \*/  {  int Member; /\* members name space\*/  } MyStructObj; /\* OK - identifiers name space\*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 5 |
| 2004-5\_7 | No identifier name should be reused |
| Description: | Example:  struct air\_speed  {  int speed;  } \* x;  struct gnd\_speed  {  int speed; /\* Violation \*/  } \* y;  Repair:  Do not use the same names for different members.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 5 |
| 2004-6\_2 | signed and unsigned char type shall be used only for the storage and use of numeric values |
| Description: | Example:  signed char a = 'A'; /\* Violation \*/  void foo()  {  a = 'B'; /\* Violation \*/  if (a == 'C') /\* Violation \*/  {  }  if (a < 'D' ) /\* Violation \*/  {  }  }  Repair:  unsigned char b = 65; /\* OK \*/  void goo()  {  a = 66; /\* OK \*/  if (a == 67) /\* OK \*/  {  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 6 |
| 2004-6\_3 | typedefs that indicate size and signedness should be used in place of the basic types |
| Description: | Example:  char a; /\* Violation \*/  float b; /\* Violation \*/  void foo()  {  double h; /\* Violation \*/  int z; /\* Violation \*/  }  Repair:  typedef char char\_t;  typedef signed char int8\_t;  typedef signed short int16\_t;  typedef signed int int32\_t;  typedef signed long int64\_t;  typedef unsigned char uint8\_t;  typedef unsigned short uint16\_t;  typedef unsigned int uint32\_t;  typedef unsigned long uint64\_t;  typedef float float32\_t;  typedef double float64\_t;  typedef long double float128\_t;  char\_t a; /\* OK \*/  float32\_t b; /\* OK \*/  void foo1()  {  float64\_t h; /\* OK \*/  int32\_t z; /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 6 |
| 2004-6\_4 | Bit fields shall only be defined to be of type unsigned int or signed int |
| Description: | Example:  enum Enum { E1, E2};  struct Struct  {  unsigned char f1:2; /\* Violation \*/  unsigned short f2:2; /\* Violation \*/  unsigned long f3:2; /\* Violation \*/  enum Enum f4:2; /\* Violation \*/  int f5:2; /\* Violation \*/  };  Repair:  struct Struct2  {  unsigned int f1:2; /\* OK \*/  unsigned int f2:2; /\* OK \*/  unsigned int f3:2; /\* OK \*/  signed int f4:2; /\* OK \*/  };  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 6 |
| 2004-6\_5 | Bit fields of type signed int shall be at least 2 bits long |
| Description: | Example:  struct MyStruct  {  signed int si01:1; /\* Violation \*/  };  Repair:  struct MyStruct1  {  signed int si02:2; /\* OK \*/  };  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 6 |
| 2004-7\_1 | Octal constants (other than zero) shall not be used |
| Description: | Example:  void foo()  {  int code1;  int code2;  code1 = 052; /\* Violation \*/  code2 = 071; /\* Violation \*/  }  Repair:  void foo1()  {  int code1;  int code2;  code1 = 42; /\* OK \*/  code2 = 57; /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 7 |
| 2004-8\_11 | The static storage class specifier shall be used in definitions and declarations of objects and functions that have internal linkage |
| Description: | Example:  unsigned short x; /\* Violation \*/  Repair:  static unsigned short x; /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 8 |
| 2004-8\_12 | When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialisation |
| Description: | Example:  extern int array2[ ]; /\* Violation \*/  Repair:  int array2[ ] = { 0, 10, 15 }; /\* OK \*/  int array1[ 10 ]; /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 8 |
| 2004-8\_2 | Whenever a function is declared or defined, its type shall be explicitly stated |
| Description: | Example:  static foo(void); /\* Non-compliant - implicit type \*/  Repair:  static int foo(void); /\* Compliant - explicit type \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 8 |
| 2004-8\_2 | Whenever an object is declared or defined, its type shall be explicitly stated |
| Description: | Example:  extern x; /\* Non-compliant - implicit int type \*/  const y; /\* Non-compliant - implicit int type \*/  Repair:  extern int16\_t x; /\* Compliant - explicit type \*/  const int16\_t y; /\* Compliant - explicit type \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 8 |
| 2004-8\_3 | Use identical types in declaration and definition |
| Description: | Example:  typedef int MyInt;  typedef MyInt MyInt2;  int foo1(); /\* Violation \*/  MyInt foo1()  {  return 0;  }  MyInt2 foo2(); /\* Violation \*/  int foo2()  {  return 0;  }  Repair:  typedef int MyInt;  typedef MyInt MyInt2;  int foo1(); /\* OK \*/  int foo1()  {  return 0;  }  MyInt2 foo2(); /\* OK \*/  MyInt2 foo2()  {  return 0;  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 8 |
| 2004-8\_3 | Use identical types in declaration and definition |
| Description: | Example:  typedef int MyInt1;  typedef MyInt1 MyInt2;  void foo1(MyInt1 a, int b); /\* Violation \*/  void foo1(int a, MyInt2 b) {}  void foo2(int a, MyInt1 b); /\* Violation \*/  void foo2(int a, MyInt2 b) {}  Repair:  typedef int MyInt1;  typedef MyInt1 MyInt2;  void foo1(MyInt1 a, int b); /\* OK \*/  void foo1(int a, MyInt1 b) {}  void foo2(int a, MyInt2 b); /\* OK \*/  void foo2(int a, MyInt2 b) {}  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 8 |
| 2004-8\_5 | There shall be no definitions of objects or functions in a header file |
| Description: | Example:  /\* file.h \*/  void foo1()  {  }  Repair:  /\* file.h \*/  void foo1();  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 8 |
| 2004-8\_6 | Always declare functions at file scope |
| Description: | Example:  void foo1()  {  void foo2(); /\* Violation \*/  }  Repair:  void foo2(); /\* OK \*/  void foo1()  {  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 8 |
| 2004-8\_7 | Objects shall be defined at block scope if they are only accessed from within a single function |
| Description: | Example:  int max=20; /\* Violation \*/  void foo()  {  int i =max;  max++;  }  Repair:  int max=20;  void foo()  {  int i =max;  max++;    }  void moo()  {  for(int i=0; i<max;i++){}  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 8 |
| 2004-9\_2 | Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures |
| Description: | Example:  int y[3][2] = { 1, 2, 3, 4, 5, 6 }; /\* Violation \*/  Repair:  int z[3][2] = { { 1, 2 }, { 3, 4 }, { 5, 6 } }; /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 9 |
| 2004-9\_3 | In an enumerator list, the ??construct shall not be used to explicitly initialise members other than the first, unless all items are explicitly initialised |
| Description: | Example:  enum TEST { /\* Violation \*/  X=5,  Y,  Z=9,  };  enum TEST2 { /\* Violation \*/  X2,  Y2=5,  Z2,  };  Repair:  enum TEST3 { /\* OK \*/  X3,  Y3,  Z3,  };  enum TEST4 { /\* OK \*/  X4=2,  Y4,  Z4,  };  enum TEST5 { /\* OK \*/  X5=2,  Y5=4,  Z5=8,  };  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 9 |
| 2004-10\_3 | The value of a complex expression of integer type may only be cast to a type that is narrower and of the same signedness as the underlying type of the expression |
| Description: | Example:  typedef char char\_t;  typedef signed char int8\_t;  typedef signed short int16\_t;  typedef signed int int32\_t;  typedef signed long int64\_t;  typedef unsigned char uint8\_t;  typedef unsigned short uint16\_t;  typedef unsigned int uint32\_t;  typedef unsigned long uint64\_t;  typedef float float32\_t;  typedef double float64\_t;  typedef long double float128\_t;  int foo(int);  void goo()  {  float64\_t f64a, f64b;  float32\_t f32a, f32b;  uint32\_t u32a, u32b;  uint16\_t u16a,u16b;  uint8\_t u8a,u8b;  int32\_t buf16a[100];  int32\_t s32a,s32b;  int16\_t s16a, s16b;  (float64\_t)(s32a / s32b); /\* Violation \*/  (float64\_t)(s32a > s32b) ; /\* Violation \*/  (float32\_t)(u16a + u16b); /\* Violation \*/  (float64\_t)foo(u16a + u16b); /\* Violation \*/  (int32\_t)(u32a + u32b); /\* Violation \*/  }  Repair:  void goo()  {  float64\_t f64a, f64b;  float32\_t f32a, f32b;  uint32\_t u32a, u32b;  uint16\_t u16a,u16b;  uint8\_t u8a,u8b;  int32\_t buf16a[100];  int32\_t s32a,s32b;  int16\_t s16a, s16b;    (float32\_t)(f64a + f64b); /\* OK - float type expression \*/  (float64\_t)(f32a + f32b); /\* OK - float type expression \*/  (float64\_t)f32a; /\* OK - not complex expression \*/  (float64\_t)s32a / (float32\_t)s32b; /\* OK - not complex expression \*/  (uint32\_t)u16a + u16b; /\* OK - not complex expression \*/  (int16\_t)(s32a - 12345); /\* OK - cast to narrow type\*/  (uint8\_t)(u16a \* u16b); /\* OK - cast to narrow type\*/  (int16\_t)(s32a \* s32b); /\* OK - cast to narrow type\*/  (uint16\_t)(f64a + f64b); /\* OK - float type expression \*/  (uint32\_t)(u16a + u16b); /\* OK - expression's result type is int32\_t \*/  (int32\_t)buf16a[u16a + u16b];/\* OK - expression's result type is int32\_t \*/  (uint16\_t)(u8a \* u8b) ; /\* OK - expression's result type is int32\_t \*/  (int32\_t)(s16a \* s16b); /\* OK - expression's result type is int32\_t \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 10 |
| 2004-10\_4 | The value of a complex expression of floating type may only be cast to a narrower floating type |
| Description: | Example:  typedef float float32\_t;  typedef double float64\_t;  typedef long double float128\_t;  void goo()  {  float128\_t f128a, f128b;  float64\_t f64a, f64b;  float32\_t f32a, f32b;  (float64\_t)(f32a + f32b); /\* Violation \*/  }  Repair:  void goo()  {  float128\_t f128a, f128b;  float64\_t f64a, f64b;  float32\_t f32a, f32b;  (float32\_t)(f64a + f64b); /\* OK - cast to narrow float type \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 10 |
| 2004-10\_5 | If the bitwise operators ~ and &lt;&lt; are applied to an operand of underlying type unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand |
| Description: | Example:  typedef unsigned char uint8\_t;  typedef unsigned short uint16\_t;  void foo()  {  uint8\_t port = 0x5aU;  uint8\_t result\_8;  uint16\_t result\_16;  uint16\_t mode;    result\_8 = (~port) >> 4; /\* Violation \*/  result\_16 = ((port << 4) & mode) >> 6; /\* Violation \*/  }  Repair:  typedef unsigned char uint8\_t;  typedef unsigned short uint16\_t;  void foo()  {  uint8\_t port = 0x5aU;  uint8\_t result\_8;  uint16\_t result\_16;  uint16\_t mode;    result\_8 = ((uint8\_t)(~port)) >> 4 ; /\* OK \*/  result\_16 = ( (uint16\_t) (~(uint16\_t)port) ) >> 4 ; /\* OK \*/  result\_16 = ((uint16\_t)((uint16\_t)port << 4) & mode) >> 6; /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 10 |
| 2004-11\_1 | Conversions shall not be performed between a pointer to a function and any type other than an integral type |
| Description: | Example:  typedef int (\*func\_t)(char \*);  typedef float (\*func\_t2)(char \*);  int foo(char \*x)  {  return 1;  }  float goo(char \*x)  {  return 1.1;  }  void moo()  {  func\_t p = foo;  func\_t2 p2 = goo;  p= (func\_t)p2; /\* Violation \*/  }  Repair:  typedef float (\*func\_t2)(char \*);  float goo(char \*x)  {  return 1.1;  }  float goo2(char \*x)  {  return 0.0;  }  void moo()  {  func\_t2 p2 = goo;  func\_t2 p3 = goo2;  p2= (func\_t2)p3; /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 11 |
| 2004-11\_2 | Conversions shall not be performed between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void |
| Description: | Example:  struct A;  void foo()  {    struct A \*ptr;  float \*arr[10];  int \*p;  ptr = p; /\* Violation \*/  }  Repair:  struct A;  void foo()  {    struct A \*ptr;  float \*arr[10];  int \*p;  p = arr; /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 11 |
| 2004-11\_3 | A cast should not be performed between a pointer type and an integral type |
| Description: | Example:  void foo( ) {  int\* pi;  int i;  i = (int) pi; // Violation  }  Repair:  Do not convert pointer to integer;  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 11 |
| 2004-11\_4 | A cast should not be performed between a pointer to object type and a different pointer to object type |
| Description: | Example:  void foo( ) {  int\* pi;  char\* i;  i = (char\*) pi; // Violation  i = (char\*) &i; // Violation  }  Repair:  Do not convert pointer to different pointer.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 11 |
| 2004-11\_5 | A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer |
| Description: | Example:  void foo( ) {  const int ci;  int i;  i = (int) ci; // Violation  i = (int) &i; // Violation  }  Repair:  Do not convert const to non const.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 11 |
| 2004-12\_1 | Limited dependence should be placed on C’s operator precedence rules in expressions |
| Description: | Example:  void foo() {  int a, b;  b = a = 0; // Violation  }  Repair:  void foo() {  int a, b;  b = (a = 0); // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_1 | Limited dependence should be placed on C’s operator precedence rules in expressions |
| Description: | Example:  void foo() {  int a, b;  b = (a = 0); // OK  b = (a + 0); // Violation  }  Repair:  void foo() {  int a, b;  b = (a = 0); // OK  b = a + 0; // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_1 | Limited dependence should be placed on C’s operator precedence rules in expressions |
| Description: | Example:  void foo( ) {  int a, b;  b = a \* (-1); // Violation  }  Repair:  void foo( ) {  int a, b;  b = a \* -1; // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_1 | Limited dependence should be placed on C’s operator precedence rules in expressions |
| Description: | Example:  void fooPlus( ) {  unsigned \_\_int16 a = 10;  unsigned \_\_int16 b = 65535;  unsigned \_\_int32 c = 0;  unsigned \_\_int32 d;  d = (a + b) + c; /\* d is 9; a + b wraps modulo 65536 \*/  }  void fooMultiply( ) {  unsigned \_\_int16 a = 10;  unsigned \_\_int16 b = 65535;  unsigned \_\_int32 c = 0;  unsigned \_\_int32 d;  d = (a \* b) \* c; /\* d is 65526; a \* b wraps modulo 65536 \*/  }  Repair:  void fooPlus( ) {  unsigned \_\_int16 a = 10;  unsigned \_\_int16 b = 65535;  unsigned \_\_int32 c = 0;  unsigned \_\_int32 d;  d = a + (b + c); /\* d is 65545 \*/  }  void fooMultiply( ) {  unsigned \_\_int16 a = 10;  unsigned \_\_int16 b = 65535;  unsigned \_\_int32 c = 0;  unsigned \_\_int32 d;  d = a \* (b \* c); /\* d is 655350 \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_10 | The comma operator shall not be used |
| Description: | Example:  void foo() {  int x, y;  x = 0, y = 0; // Violation  }  Repair:  void foo() {  int x, y;  x = 0;  y = 0; // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_12 | The underlying bit representations of floating-point values shall not be used |
| Description: | Example:  struct S {  float f:6; // Violation  };  union U {  float f:6; // Violation  };  Repair:  struct S {  int f:6; // OK  };  union U {  int f:6; // OK  };  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_13 | The increment (++) and decrement (--) operators should not be mixed with other operators in an expression |
| Description: | Example:  void foo() {  int x, y;  x = --y + x++; /\* Violation \*/  }  Repair:  void foo() {  int x, y;  --y; /\* OK \*/  x = y + x;  x++; /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_2 | The value of an expression shall be the same under any order of evaluation that the standard permits |
| Description: | Example:  void foo( ) {  int a, b[10];  a = b[a] + a++; // Violation  }  Repair:  void foo( ) {  int a, b[10];  a = b[a] + a; // OK  a++; // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_2 | The value of an expression shall be the same under any order of evaluation that the standard permits |
| Description: | Example:  int foo( int i, int j ) {  j = func( i++, i ); // Violation  return 0;  }  Repair:  int foo( int i, int j ) {  j = func( i, i ); // OK  i++;  return 0;  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_2 | The value of an expression shall be the same under any order of evaluation that the standard permits |
| Description: | Example:  struct S {  void (\*task\_start\_fn)( struct S\* );  };  void foo() {  struct S\* p;  p->task\_start\_fn( p++ ); // Violation  }  Repair:  struct S {  void (\*task\_start\_fn)( struct S\* );  };  void foo() {  struct S\* p;  p->task\_start\_fn( p ); // OK  p++;  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_2 | The value of an expression shall be the same under any order of evaluation that the standard permits |
| Description: | Example:  int foo( int x ) {  if (x < 0)  return 0;  return foo( x - 1 ) - foo( x - 2 ); // Violation  }  Repair:  int foo( int x ) {  int y;  if (x < 0)  return 0;  y = foo( x - 1 );  return y - foo( x - 2 ); // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_2 | The value of an expression shall be the same under any order of evaluation that the standard permits |
| Description: | Example:  void foo( int x, int y, int z ) {  x = y = z / 3; // Violation  }  Repair:  void foo( int x, int y, int z ) {  y = z / 3; // OK  x = y; // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_3 | The sizeof operator shall not be used on expressions that contain side effects |
| Description: | Example:  void foo( ) {  int i;  int j;  j = sizeof( i = 1234 ); // Violation  }  Repair:  void foo( ) {  int i;  int j;  i = 1234;  j = sizeof( i ); // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_4 | The right-hand operand of a logical && or || operator shall not contain side effects |
| Description: | Example:  void foo( ) {  int i;  int j;  if ((j == i) || (0 == i++)) ; /\* Violation \*/  }  Repair:  void foo( ) {  int i;  int j;  if ((j == i) || (0 == i)) i++; /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_4 | The right-hand operand of a logical && or || operator shall not contain side effects |
| Description: | Example:  void foo( ) {  int i;  int j;  (i > j)? j++: i++; // Violation  }  Repair:  void foo( ) {  int i;  int j;  if (i > j) {  j++; // OK  } else {  i++; // OK  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_5 | The operands of a logical && or || shall be primary-expressions |
| Description: | Example:  int foo( int x, int y, int z ) {  if ( x || y && z ); // Violation  if ( x && !y ); // Violation  if ( foo( x, y, z ) && x ); // Violation  return 0;  }  Repair:  int foo( int x, int y, int z ) {  if ( x || ( y && z ) ); // OK  if ( x && ( !y ) ); // OK  if ( ( foo( x, y, z ) ) && x ); // OK  return 0;  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_6 | The operands of logical operators (&&, || and !) should be effectively Boolean |
| Description: | Example:  float goo();  void foo() {  float x;  int y, z;  z = (x > y) && goo(); // Violation  z = !(x = y); // Violation  if (goo()); // Violation  }  Repair:  float goo();  void foo() {  float x;  int y;  z = (x > y) && (goo() != 0); // OK  z = !(x == y); // OK  if (goo() != 0); // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_7 | Bitwise operators shall not be applied to operands whose underlying type is signed |
| Description: | Example:  int foo( int x, int y ) {  if (x == 0)  return ~x; // Violation  if (x == 0)  return x << 2; // Violation  if (x == 0)  return x >> 2; // Violation  if (x == 0)  return x & 0x0F; // Violation  if (x == 0)  return x ^ 0xF0; // Violation  return x | 0xF0; // Violation  }  Repair:  int foo( unsigned int x, unsigned int y ) {  if (x == 0)  return ~x; // OK  if (x == 0)  return x << 2; // OK  if (x == 0)  return x >> 2; // OK  if (x == 0)  return x & 0x0F; // OK  if (x == 0)  return x ^ 0xF0; // OK  return x | 0xF0; // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_8 | The right-hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left-hand operand |
| Description: | Example:  #ifndef uint8\_t  typedef unsigned char uint8\_t;  #endif  #ifndef uint16\_t  typedef unsigned short uint16\_t;  #endif  #ifndef uint32\_t  typedef unsigned int uint32\_t;  #endif  void foo( ) {  uint8\_t it8;  uint16\_t it16;  uint32\_t it32;  i8 << 8; // Violation  i16 << 16; // Violation  i32 >> 32; // Violation  i16 >> i8; // Violation  }  Repair:  #ifndef uint8\_t  typedef unsigned char uint8\_t;  #endif  #ifndef uint16\_t  typedef unsigned short uint16\_t;  #endif  #ifndef uint32\_t  typedef unsigned int uint32\_t;  #endif  void foo( ) {  uint8\_t it8;  uint16\_t it16;  uint32\_t it32;  i8 << 7; // OK  i16 << 15; // OK  i32 >> 31; // OK  if (i8 > 0 && i8 < sizeof( i8 ))  i16 >> i8; // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-12\_9 | The unary minus operator shall not be applied to an expression whose underlying type is unsigned |
| Description: | Example:  void foo() {  unsigned char ui1;  signed short si2;  si2 = -ui1; // Violation  }  Repair:  void foo() {  unsigned char ui1;  signed short si2;  si2 = -(signed short) ui1; // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 12 |
| 2004-13\_1 | Assignment operators shall not be used in expressions that yield a Boolean value |
| Description: | Example:  void foo() {  int x;  int y;  int z;  z = !(x = y); // Violation  if ((x > y) && (x = 4)); // Violation  if (!(x = y)); // Violation  }  Repair:  void foo() {  int x;  int y;  int z;  z = !(x == y); // OK  if ((x > y) && (x == 4)); // OK  if (!(x == y)); // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 13 |
| 2004-13\_2 | Tests of a value against zero should be made explicit, unless the operand is effectively Boolean |
| Description: | Example:  float goo();  void foo() {  float x;  int y, z;  z = (x > y) && goo(); // Violation  z = !(x = y); // Violation  if (goo()); // Violation  }  Repair:  float goo();  void foo() {  float x;  int y;  z = (x > y) && (goo() != 0); // OK  z = !(x == y); // OK  if (goo() != 0); // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 13 |
| 2004-13\_3 | Floating-point expressions shall not be tested for equality or inequality |
| Description: | Example:  void foo() {  float x, y;  if (x == y); // Violation  if (x == 0.0f); // Violation  }  Repair:  void foo( float epsilon ) {  float x, y;  if (x - epsilon <= y && y <= x + epsilon); // OK  if (-epsilon <= x && x <= epsilon); // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 13 |
| 2004-13\_4 | The controlling expression of a for statement shall not contain any objects of floating type |
| Description: | Example:  void foo( float x ) {  float y;  for (y = x; y < x + 10; y = y + 1); // Violation  }  Repair:  void foo( float x ) {  int y, z;  for (z = x, y = x; y < z + 10; z++); // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 13 |
| 2004-13\_5 | The three expressions of a for statement shall be concerned only with loop control |
| Description: | Example:  void foo( ) {  int x, j;  for (x += 5; x; x += 2); // Violation  for (x = 5; x += 2;x++); // Violation  for (x = 5; x < 0;x += 2); // Violation  for (x = j; x < j; j++); // Violation  }  Repair:  void foo( ) {  int x, j;    for (x = x + 5; x; x++, x++); // OK  for (x = 5; x > 0; x++, x++, x++); // OK  for (x = 5; x < 0; x++, x++); // OK  for (x = j; x < j; x++); // OK  for ( ; x += 2; x += 2); // OK  for (x += j; x < j++ ; ); // OK  for ( ; ; ); // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 13 |
| 2004-13\_6 | Do not modify for loop counter within a body of the loop |
| Description: | Example:  void foo()  {  int i;  for ( i = 0; i < 5; i++ )  {  i = i + 3; /\* Violation \*/  }  }  Repair:  void foo1()  {  int i;  for ( i = 0; i < 5; i = i + 3 ) /\* OK \*/  {  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 13 |
| 2004-14\_1 | There shall be no unreachable code |
| Description: | Example:  void foo()  {  int i = 0;  if(1)  {  i = 1;  } else { /\* Violation \*/  i = 2;  }  }  Repair:  void foo1()  {  int i = 0;  if(i==1) /\* OK \*/  {  i = 1;  } else {  i = 2;  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 14 |
| 2004-14\_1 | There shall be no unreachable code |
| Description: | Example:  int foo()  {  int i = 0;  return(i);  i = 7; /\* Violation \*/  }  int foo1()  {  int i = 0;  return(  i  );  i=6; /\* Violation \*/  }  void foo2(int event)  {  int a;  switch (event)  {  case 1:  break;  a = 7; /\* Violation \*/  default:  break;  }  }  Repair:  int foo3()  {  int i = 0;  return(i); /\* OK \*/  }  int foo4()  {  int i = 0;  switch(i){  case 1 : return(i); /\* OK \*/  case 2 : return(i); /\* OK \*/  case 3 : return(i); /\* OK \*/  }  return(i); /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 14 |
| 2004-14\_10 | All if ... else if constructs shall be terminated with an else clause |
| Description: | Example:  void foo(int a)  {  if(a > 0)  {  }  else if (a > 10) /\* Violation \*/  {  }  }  Repair:  void goo(int a)  {  if(a > 0)  {  }  else if (a > 10) /\* OK \*/  {  }  else  {  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 14 |
| 2004-14\_2 | All non-null statements shall either have at least one side-effect however executed or cause control flow to change |
| Description: | Example:  struct tagStruct  {  int \_iField;  char \* \_p;  };  volatile struct tagStruct \_SSS;  int bar();  void foo()  {  int i = 0;  volatile int j = 0;  struct tagStruct s, \*ps;  /\* no side effects: \*/  i + 3; /\* Violation - result not used \*/  3; /\* Violation - result not used \*/  i; /\* Violation - result not used \*/  i + bar(); /\* Violation - result not used \*/  switch (i) { } /\* Violation - empty block - no side-effect \*/  ps->\_p + s.\_iField; /\* Violation - result not used \*/  ps->\_iField << s.\_iField; /\* Violation - result not used \*/  \*(ps->\_p); /\* Violation - result not used \*/    { /\* Violation - empty block - no side-effect \*/  }  }  Repair:  struct tagStructOK  {  int \_iField;  char \* \_p;  };  volatile struct tagStructOK \_SSSOK;  int bar();  void fooOK()  {  int i = 0;  volatile int j = 0;  struct tagStructOK s, \*ps;    /\* with side effects: \*/  i = i + 3; /\* OK - assignment \*/  i <<= 3; /\* OK - assignment \*/  bar(); /\* OK - function call \*/  j; /\* OK - volatile variable \*/  \_SSSOK.\_p; /\* OK - volatile variable \*/    /\* OK - null statement \*/  ;  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical system  Chapter 6, Section 14 |
| 2004-14\_3 | Null statements format |
| Description: | Example:  void foo()  {  /\* Violation \*/ ;  ;/\* Violation \*/  }  Repair:  void goo()  {  /\* OK \*/  ;  ; /\* OK \*/  ; /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 14 |
| 2004-14\_3 | Before preprocessing, a null statement shall occur only on a line by itself |
| Description: | Example:  void foo(int a)  {  ;; /\* Violation \*/    a = 8;;/\* Violation \*/  ; a=7; /\* Violation \*/  ;{ /\* Violation \*/  }  }  Repair:  void goo()  {  /\* OK \*/  ;  ; /\* OK \*/  ; /\* OK \*/  }; /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 14 |
| 2004-14\_4 | The goto statement shall not be used |
| Description: | Example:  void func(int a)  {  if (a==1)  {  goto end; /\* Violation \*/  }  end: return;  }  Repair:  Do not use the goto statement in program code.      References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 14 |
| 2004-14\_5 | The continue statement shall not be used |
| Description: | Example:  int bar(int);  void func(int i)  {  while(i--)  {  if (bar(i))  {  continue; /\* Violation \*/  }  }  }  Repair:  Avoid using 'continue' in program source.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 14 |
| 2004-14\_6 | For any iteration statement there shall be at most one break statement used for loop termination |
| Description: | Example:  void foo()  {  int a;  for(a = 0; a <10; a++) /\* Violation \*/  {  if ( a == 5)  {  break;  }  if ( a == 7)  {  break;  }  }  }  Repair:  void goo()  {  int a;  for(a = 0; a <10; a++) /\* OK \*/  {  if ( a == 5)  {  }  }  for(a = 0; a <10; a++) /\* OK \*/  {  if ( a == 5)  {  break;  }  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 14 |
| 2004-14\_7 | A function shall have a single point of exit at the end of the function |
| Description: | Example:  int foo1(int a) { /\* Violation - two returns in function \*/  if (a>0) {  return 1;  } else {  return 0;  }  }  int foo2(int a) {  int result;  if (a>0) {  return result; /\* Violation - 'return' not at the end of function \*/  }  }  Repair:  int foo3(int a) { /\* OK \*/  int result;  if (a>0) {  result=1;  } else {  result=0;  }    return result;  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 14 |
| 2004-14\_8 | The statement forming the body of a 'switch', 'while', 'do...while' or 'for' statement shall be a compound statement |
| Description: | Example:  void foo(int x)  {  int i;  switch(i) /\* Violation \*/  ;  for (i=0; i< 10; i++)/\* Violation \*/  foo( x );  while (1) /\* Violation \*/  foo( x );  do /\* Violation \*/  foo( x );  while(1);  }  Repair:  void foo2(int x)  {  int i;  switch(i) /\* OK \*/  {  }  for (i=0; i< 10; i++)/\* OK \*/  {foo( x );}  while (1) /\* OK \*/  {foo( x );}  do /\* OK \*/  {foo( x );}  while(1);  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 14 |
| 2004-14\_9 | 'if and 'else' construct should be followed by a compound statement |
| Description: | Example:  void foo(int x)  {  int i;  if (x>0) /\* Violation \*/  foo( x );  if (x>0) /\* Violation \*/  foo( x );  else if (x < -1) {}  if (x >0) {  } else /\* Violation \*/  foo( x );  if (i >5 )  {  } else /\* Violation \*/  ;  }  Repair:  void foo2(int x)  {  int i;  if (x>0) /\* OK \*/  {foo( x );}  if (x>0) /\* OK \*/  {foo( x );}  else if (x < -1) {}  if (x >0) { /\* OK \*/  } else  {foo( x );}  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 14 |
| 2004-15\_0 | A switch statement shall only contain switch labels and switch clauses, and no other code |
| Description: | Example:  void foo(int a,int b)  {  int c;  switch(a)  {    case 1:  break;    c = 1; /\* Violation \*/    case 2:  break;    c = 2; /\* Violation \*/    default:  break;    c = 3; /\* Violation \*/  }  }  Repair:  void foo1(int a,int b)  {  int c;  switch(a)  {    case 1:  c = 1; /\* OK \*/  break;    case 2:  c = 2; /\* OK \*/  break;    default:  c = 3; /\* OK \*/  break;  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 15 |
| 2004-15\_0 | A switch statement shall only contain switch labels and switch clauses, and no other code |
| Description: | Example:  void foo(int a,int b)  {  int c;  switch(a)  {  c = 0; /\* Violation \*/  case 1:  break;  case 2:  break;    default:  break;  }  }  Repair:  void foo1(int a,int b)  {  int c;  c = 0; /\* OK \*/  switch(a)  {  case 1:  break;  case 2:  break;    default:  break;  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 15 |
| 2004-15\_1 | A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement |
| Description: | Example:  void foo(int a,int b)  {  switch(a)  {    case 1:  {  case 6: /\* Violation \*/  ;  default: /\* Violation \*/    break;  }  break;  case 2:  if (b==1)  {  case 3: /\* Violation \*/  break;  }  break;  }  }  Repair:  Do not use nested 'case'/'default' statements.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 15 |
| 2004-15\_2 | An unconditional break statement shall terminate every non-empty switch clause |
| Description: | Example:  void foo(int i) {  switch(i)  {  case 0 : /\* Violation \*/  i++;  case 1 : /\* Violation \*/  i++;  if (i>0) {  i = 5;  break;  }  case 4 :  break;  case 5: /\* Violation \*/  i++;  default:  ;  }  }  Repair:  void foo2(int i) {  switch(i)  {  case 0 : /\* OK \*/  i++;  break;  case 1 : /\* OK \*/  i++;  if (i>0) {  i = 5;  }  break;  case 3 : /\* OK - empty case \*/  default:  ;  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 15 |
| 2004-15\_3 | The final clause of a switch statement shall be the default clause |
| Description: | Example:  void foo(int i) {  switch(i) /\* Violation \*/  {  case 0 : break;  case 1 : break;  }  switch(i) /\* Violation \*/  {  case 0 : break;  default: break;  case 1 : break;  }  }  Repair:  void foo(int i) {  switch(i) /\* OK \*/  {  case 0 : break;  case 1 : break;  default: break;  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 15 |
| 2004-15\_4 | A switch expression shall not represent a value that is effectively Boolean |
| Description: | Example:  void foo(int i) {  switch(i == 0) /\* Violation \*/  {  case 0 : break;  default:;  }  }  Repair:  void foo1(int i) {  switch(i) /\* OK \*/  {  case 0 : break;  default:;  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 15 |
| 2004-15\_5 | Every switch statement shall have at least one case clause |
| Description: | Example:  void foo(int i)  {  switch(i) /\* Violation \*/  {  default:  ;  }  }  Repair:  void foo(int i)  {  switch(i) /\* OK \*/  {  case 1:  {  }  default:  ;  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 15 |
| 2004-16\_1 | Functions shall not be defined with a variable number of arguments |
| Description: | Example:  #include <stdio.h>  #define ANSI  #ifdef ANSI  #include <stdarg.h> /\* Violation \*/  int average( int first, ... );  int average( int first, ... )  {  int count = 0, sum = 0, i = first;  va\_list marker;  va\_start( marker, first ); /\* Violation \*/  while( i != -1 )  {  sum += i;  count++;  i = va\_arg( marker, int); /\* Violation \*/  }  va\_end( marker ); /\* Violation \*/  return( sum ? (sum / count) : 0 );  }  Repair:  Do not use functions with variable numbers of arguments.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 16 |
| 2004-16\_10 | If a function returns error information, then that error information shall be tested |
| Description: | Example:  int SomeFunctionReturningError( );  void foo( ) {  SomeFunctionReturningError( ); // Violation  }  Repair:  int SomeFunctionReturningError( );  int foo( ) {  int x;  x = SomeFunctionReturningError( ); // OK  if (SomeFunctionReturningError( )); // OK  switch (SomeFunctionReturningError( )) { // OK  }  return SomeFunctionReturningError( ); // OK  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 16 |
| 2004-16\_2 | Functions shall not call themselves, either directly or indirectly |
| Description: | Example:  void foo1(int l)  {  int x=l;  if (l > 0) {  foo1(x-1); /\* Violation \*/  }  }  Repair:  void foo2() { /\* OK \*/  /\* empty \*/  }  void foo3(); /\* OK \*/  void foo5(int);  void foo4(int l) {  int x=l;  if (l > 0) {  foo5(x-1); /\* OK - cannot check indirect recursion \*/  }  }  void foo5(int l) {  int x=l;  if (l > 0) {  foo4(x-1); /\* OK - cannot check indirect recursion \*/  }  }  void foo6() {  foo2(); /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 16 |
| 2004-16\_3 | Identifiers shall be given for all of the parameters in a function prototype declaration |
| Description: | Example:  void foo3(int a, int); /\* Violation \*/  Repair:  void foo(); /\* OK \*/  void foo2(int a, int b); /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 16 |
| 2004-16\_4 | The identifiers used in the declaration and definition of a function shall be identical |
| Description: | Example:  void foo(int a, int b);  void foo(int x, int y ) {} /\* Violation \*/  Repair:  void foo(int a, int b); /\* OK \*/  void foo(int a, int b) {}  void foo1(); /\* OK \*/  void foo2(int , int ); /\* OK \*/  void foo2(int x, int y ) {}  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 16 |
| 2004-16\_5 | Functions with no parameters shall be declared with parameter type void |
| Description: | Example:  void foo(); /\* Violation \*/  Repair:  void foo(void); /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 16 |
| 2004-16\_6 | The number of arguments passed to a function shall match the number of parameters |
| Description: | Example:  void foo0( void );  void foo3( int, int, int );  void goo( ) {  foo0( 7 ); /\* Violation \*/  foo3( 6, 6, 6, 6); /\* Violation \*/  }  Repair:  void foo0( void );  void foo2( int, int, ... );  void foo3( int, int, int );  void goo( ) {  foo0( ); /\* OK \*/  foo2( 6, 6 ); /\* OK \*/  foo2( 6, 6, 6, 6 ); /\* OK \*/  foo3( 6, 6, 6 ); /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 16 |
| 2004-16\_7 | A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not used to modify the addressed object |
| Description: | Example:  void foo1( /\* Violation - z should be const\*/  int x,  int\* y,  int\* z  )  {  \*y = \*z;  }  Repair:  void foo2( /\* OK \*/  int x,  int\* y,  const int\* z  )  {  \*y = \*z;  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 16 |
| 2004-16\_8 | All exit paths from a function with non-void return type shall have an explicit return statement with an expression |
| Description: | Example:  int foo1(int x){ // Violation  // in second if statement, return statement is missing  if (x==0) {  if (x==0) {  } else {  return 0;  }  } else {  return 0;  }  }  int foo2(int x){ // Violation  // in switch statement, default statement is missing  switch(x){  case 0: return 1;  case 1: return 1;  case 2: return 1;  }  }  int foo3(int x){ // Violation  }  Repair:  int foo1(int x){ // OK  if (x==0) {  if (x==0) {  return 0;  } else {  return 0;  }  } else {  return 0;  }  }  int foo2(int x){ // OK  switch(x){  case 0: return 1;  case 1: return 1;  case 2: return 1;  default: return 1;  }  }  int foo3(int x){ // OK  return 0;  }  int foo4(int x){ // OK  if (x==0) {  return 0;  }  return 0;  }  int foo5(int x){ // OK  if (x==0) {  if (x==0) {  return 0;  } else {  return 0;  }  } else {  switch(x){  case 0: return 1;  case 1: return 1;  case 2: return 1;  default: return 1;  }  }  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 16 |
| 2004-17\_1 | Pointer arithmetic shall only be applied to pointers that address an array or array element |
| Description: | Example:  void foo( int a[] ) {  int\* p1 = 0;  int\* p2;  int\* p3 = a;  a++; // OK  p1++; // Violation  p2 = a;  p2++; // OK  p3++; // OK  }  Repair:  Do not apply pointer arithmetic to pointers.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 17 |
| 2004-17\_2 | Pointer arithmetic shall only be applied to pointers that address an array or array element |
| Description: | Example:  void foo( int a[] ) {  int\* p1 = 0;  int\* p2;  int\* p3 = a;  int i;  i = p1 - p2; // Violation  i = p2 - a; // Violation  i = p3 - a; // OK  }  Repair:  Do not apply pointer subtraction to pointers that address elements of not  the same array.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 17 |
| 2004-17\_3 | &gt;, &gt;=, &lt;, &lt;= shall not be applied to pointer types except where they point to the same array |
| Description: | Example:  void foo( int a[] ) {  int\* p1 = 0;  int\* p2;  int\* p3 = a;  int i;  i = p1 < p2; // Violation  i = p2 < a; // Violation  i = p3 < a; // OK  }  Repair:  Do not apply pointer comparision to pointers that address elements of not  the same array.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 17 |
| 2004-17\_4 | Array indexing shall be the only allowed form of pointer arithmetic |
| Description: | Example:  void my\_fn(int \* p1, int p2[]) {  int index = 0;  int \* p3;  int \* p4;  \*p1 = 0;  p1 ++; /\* Violation - pointer increment \*/  p1 = p1 + 5; /\* Violation - pointer increment \*/  p1[5] = 0; /\* Violation - p1 was not declared as an array \*/  p3 = &p1[5]; /\* Violation - p1 was not declared as an array \*/  p2[0] = 0;  index ++;  index = index + 5;  p2[index] = 0; /\* OK \*/  p4 = &p2[5]; /\* OK \*/  }  void foo() {  int a1[16];  int a2[16];  int a[10];  int \* p;  my\_fn(a1, a2);  my\_fn(&a1[4], &a2[4]);  p = a;  \*(p+5) = 0; /\* Violation \*/  p[5] = 0; /\* OK \*/  }  Repair:  Do not apply array indexing to pointers that does not point to array elements.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 17 |
| 2004-17\_5 | The declaration of objects should contain no more than 2 levels of pointer indirection |
| Description: | Example:  typedef int \* INTPTR;  struct s {  int \*\*\* s3; /\* Violation \*/  };  struct s \*\*\* ps3; /\* Violation \*/  int \*\* (\*\*\*pfunc3)(); /\* Violation \*/  int \*\*\* ( \*\*pfunc4)(); /\* Violation \*/  void function( int \* par1,  int \*\*\* par3, /\* Violation \*/  INTPTR \* const \* const par5 /\* Violation \*/  )  {  int \*\*\* ptr3; /\* Violation \*/  INTPTR \* const \* const ptr5; /\* Violation \*/  }  Repair:  Do not declare more than two levels of pointer indirection.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 17 |
| 2004-17\_5 | The declaration of objects should contain no more than 2 levels of pointer indirection |
| Description: | Example:  typedef int\* apInt[] ;  apInt\* rule12; /\* Violation \*/  Repair:  Do not declare more than two levels of pointer indirection.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 17 |
| 2004-17\_5 | The declaration of objects should contain no more than 2 levels of pointer indirection |
| Description: | Example:  int\*\*\* (\*rule13)(); /\* Violation \*/  Repair:  Do not declare more than two levels of pointer indirection.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 17 |
| 2004-17\_5 | The declaration of objects should contain no more than 2 levels of pointer indirection |
| Description: | Example:  typedef int \* INTPTR;  void function(int \*\* par7[]) /\* Violation \*/  {  }  int \*\*\* ( \*xx5[])(); /\* Violation \*/  typedef int INTARR[];  INTARR\* ( \*\*xx9[])(); /\* Violation \*/  int\*\* rule21[]; /\* Violation \*/  Repair:  Do not declare more than two levels of pointer indirection.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 17 |
| 2004-17\_6 | The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist |
| Description: | Example:  int\* global;  int\* foo() {  int iLocal;  static int\* siLocal;  siLocal = &iLocal; // Violation  global = &iLocal; // Violation  return &iLocal; // Violation  }  void goo() {  int\* piLocal;  {  int iiLocal;  piLocal = &iiLocal; // Violation  }  }  Repair:  Do not assign local address of object to more global,  or static object or return from function.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 17 |
| 2004-18\_1 | All structure and union types shall be complete at the end of a translation unit |
| Description: | Example:  struct tnode \* pt; /\* Violation \*/  Repair:  struct tnode  {  int count;  struct tnode \* left;  struct tnode \* right;  };  struct tnode \* pt; /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 18 |
| 2004-18\_2 | An object shall not be assigned to an overlapping object |
| Description: | Example:  union U  {  int iValue;  long lValue;  double dValue;  };  void main()  {  union U a, b;  union U \*p;    a.dValue = a.lValue; /\* Violation \*/  p->dValue = p->iValue; /\* Violation \*/  }  Repair:  union U  {  int iValue;  long lValue;  double dValue;  };  void main()  {  union U a, b;  union U \*p;  a.dValue = b.dValue; /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 18 |
| 2004-18\_4 | Unions shall not be used |
| Description: | Example:  union U1 /\* Violation \*/  {  int \_i;  char \_buf[sizeof(int)];  };  Repair:  Do not use union.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 18 |
| 2004-19\_1 | #include statements in a file should only be preceded by other preprocessor directives or comments |
| Description: | Example:  void foo();  int g;  #include "test.h"  void main() {  }  Repair:  #include "test.h"  void foo();  int g;  void main() {  }    References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_10 | In the definition of a function-like macro each instance of a parameter shall be enclosed in parentheses unless it is used as the operand of # or ## |
| Description: | Example:  #define MULTI\_2(x) (x\*2) /\* Violation \*/  Repair:  #define MULTI\_2(x) ((x)\*2) /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_11 | All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator |
| Description: | Example:  #if X  /\* Violation - X undefined at this point \*/  #endif  #if Y  /\* Violation - Y undefined at this point \*/  #endif  #if X + Y  /\* Violation - X and Y undefined at this point \*/  #endif  Repair:  #define X 1  #if X  /\* OK \*/  #endif  #ifdef Y  #if Y  /\* OK - check is done in the #ifdef above \*/  #endif  #endif  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_12 | There shall be at most one occurrence of the # or ## preprocessor operators in a single macro definition |
| Description: | Example:  /\* Violation \*/  #define TEST3(A,B,C) A # B # C  /\* Violation \*/  #define TEST4(A,B,C) A ## B # C  /\* Violation \*/  #define TEST5(A,B,C) A ## B ## C  Repair:  /\* OK \*/  #define TEST(A,B) A # B  /\* OK \*/  #define TEST2(A,B) A ## B  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_13 | The # and ## preprocessor operators should not be used |
| Description: | Example:  #define stringer( x ) printf( #x "\n" ) /\* Violation \*/  Repair:  Do not use # and ## preprocessor operators.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_14 | The defined preprocessor operator shall only be used in one of the two standard forms |
| Description: | Example:  #if defined(X > Y) /\* Violation \*/  #define DEFINED defined /\* Violation \*/  #if DEFINED(X)  Repair:  #if defined(CREDIT) /\* OK \*/  foo();  #elif defined(DEBIT) /\* OK \*/  foo();  #else  goo();  #endif  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_15 | Precautions shall be taken in order to prevent the contents of a header file being included twice |
| Description: | Example:  /\* foo.h \*/  /\* Violation - file contains no multi inclusion mechanism \*/  Repair:  /\* OK \*/  #ifndef FOO\_H  #define FOO\_H  /\* code here \*/  #endif  /\* OK \*/  #ifdef FOO\_H  #error Header file is already included  #else  #define FOO\_H  #endif  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_16 | Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor |
| Description: | Example:  #define MAX 2  int foo(void)  {  int x = 0;    #ifndef MAX  x = 1;  #else1 /\* Violation \*/  x = MAX;  #endif  return x;  }  Repair:  #define MAX 2  int foo(void)  {  int x = 0;    #ifndef MAX  x = 1;  #else /\* OK \*/  x = MAX;  #endif  return x;  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_17 | All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related |
| Description: | Example:  /\* Violation \*/  #define A  #ifdef A  #include "file1.h"  #endif  #if 1  #include "file2.h"  Repair:  /\* OK \*/  #define A  #ifdef A  #include "file1.h"  #endif  #if 1  #include "file2.h"  #endif  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_2 | Non-standard characters should not occur in header file names in #include directives |
| Description: | Example:  /\* incl/header.h:\*/  #include <sys\types.h> /\* Violation \*/  #include "incl\header.h" /\* Violation \*/  Repair:  /\* incl/header.h: \*/  #include <sys/types.h> /\* OK \*/  #include "incl/header.h" /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_4 | Avoid keywords and basic types in macros |
| Description: | Example:  #define SI\_32 long /\* Violation \*/  #define STARTIF if( /\* Violation \*/  Repair:  #define PI 3.1416 /\* OK \*/  #define PLUS2(X) ((X)+2) /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_5 | Macros shall not be #define’d or #undef’d within a block |
| Description: | Example:  void foo(int\* x)  {  #define CHECKPARAM(p) (p != 0) /\* Violation \*/  if (CHECKPARAM(x)) {  /\* ... \*/  }  #undef SUM /\* Violation \*/  }  Repair:  #define SUM(A,B) ((A)+(B)) /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_6 | #undef shall not be used |
| Description: | Example:  #undef TEST /\* Violation \*/  Repair:  Do not use #undef.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_7 | A function should be used in preference to a function-like macro |
| Description: | Example:  #define SUM(A,B) ((A)+(B)) /\* Violation \*/  Repair:  int sum(int a, int b) /\* OK \*/  {  return (a+b);  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_8 | A function-like macro shall not be invoked without all of its arguments |
| Description: | Example:  #define MAX(a, b, c) ((a < b) ? b : a)  void foo(int a, int b)  {  int c;  c = MAX(a, b); /\* Violation \*/  }  Repair:  #define MAX(a, b, c) ((a < b) ? b : a)  void foo(int a, int b)  {  int c;  c = MAX(a,b, 0); /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-19\_9 | Arguments to a function-like macro shall not contain tokens that look like preprocessing directives |
| Description: | Example:  #define MACRO1(x)  #define MACRO2(x, y)  void violation(void)  {  int i = 0;  MACRO1(#foo); /\* Violation \*/  MACRO2(i, #foo); /\* Violation \*/  MACRO2(i, "#foo"); /\* Violation \*/  }  Repair:  #define MACRO1(x)  #define MACRO2(x, y)  void OK(void)  {  int i = 0;  MACRO1(i); /\* OK \*/  MACRO2(i, i); /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 19 |
| 2004-20\_1 | Reserved identifiers, macros and functions in the standard library, shall not be defined, redefined or undefined |
| Description: | Example:  #undef assert /\* Violation \*/  #if defined NDEBUG  #define assert(test) (void)0 /\* Violation \*/  #else  #define assert(test) <void expression> /\* Violation \*/  #endif  Repair:  Do not #define or #undef reserved identifiers.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |
| 2004-20\_10 | The library functions atof, atoi and atol from library stdlib.h shall not be used |
| Description: | Example:  #include <stdlib.h>  void foo( void )  {  char \*s; double x; int i; long l;  s = " -2309.12E-15";  x = atof( s ); /\* Violation \*/  s = " -9885 pigs";  i = atoi( s ); /\* Violation \*/  s = "98854 dollars";  l = atol( s ); /\* Violation \*/  }  Repair:  Do not use atof, atoi and atol functions.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |
| 2004-20\_11 | The library functions abort, exit, getenv and system from library stdlib.h shall not be used |
| Description: | Example:  #include <stdlib.h>  void foo( void )  {  char \*libvar;  libvar = getenv( "LIB" ); /\* Violation \*/  system("dir"); /\* Violation \*/  abort(); /\* Violation \*/  exit(0); /\* Violation \*/  }  Repair:  Do not use abort, exit, getenv and system functions.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |
| 2004-20\_12 | The time handling functions of library time.h shall not be used |
| Description: | Example:  #include <time.h> /\* Violation \*/  Repair:  Do not include time.h header.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |
| 2004-20\_2 | The names of standard library macros, objects and functions shall not be reused |
| Description: | Example:  void \_Assert(); /\* Violation \*/  Repair:  void foo(); /\* OK \*/  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |
| 2004-20\_3 | The validity of values passed to library functions shall be checked |
| Description: | Example:  void\* memcpy(void \*, void\*, int n);  void test(void \*v)  {  }  int getSomeIntValue()  {  return 0;  }  void foo()  {  void \*p, \*q;  int n;  memcpy(p, q, n); /\* Violation - complain about p, q and n \*/    test(q); /\* consider it a validation of q \*/    memcpy(p, q, n); /\* Violation - complain about p, and n \*/    switch (n) { /\* consider it a validation of n \*/  case 0:  break;  }    if (p == 0) { /\* consider it a validation of p \*/  }    /\* Violation - complain only about getSomeIntValue call \*/  memcpy(p, q, getSomeIntValue());  }  Repair:  void\* memcpy(void \*, void\*, int n);  void test(void \*v)  {  }  int getSomeIntValue()  {  return 0;  }  void foo()  {  void \*p, \*q;  int n;    test(q); /\* consider it a validation of q \*/    switch (n) /\* consider it a validation of n \*/  {  case 0:  break;  }    if (p == 0) /\* consider it a validation of p \*/  {  }    memcpy(p, q, n); /\* OK \*/  }  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |
| 2004-20\_4 | Dynamic heap memory allocation shall not be used |
| Description: | Example:  #include <malloc.h>  void foo()  {  char \*string;  char\* string2;  string = (char \*)malloc(10); /\* Violation \*/  string2 = (char\*)realloc(string,12); /\* Violation \*/  free(string); /\* Violation \*/  free(string2); /\* Violation \*/  }  Repair:  Do not use calloc, malloc, realloc and free functions.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |
| 2004-20\_5 | The error indicator errno shall not be used |
| Description: | Example:  #include <errno.h>  int err\_check()  {  errno = 1;  return (errno);  }  Repair:  Do not use errno.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |
| 2004-20\_6 | The macro offsetof, in library stddef.h, shall not be used |
| Description: | Example:  // Violation - in line [line\_number] is 'offsetof' macro used  #include <stddef.h>  struct S  {  int x, y, z;  char buffer [128];  };  int main()  {  int i= offsetof(struct S, buffer);  return 0;  }  Repair:  Do not use offsetof.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |
| 2004-20\_7 | The setjmp macro and the longjmp function shall not be used |
| Description: | Example:  #include <setjmp.h>  jmp\_buf mark; /\* Address for long jump to jump to \*/  int fperr; /\* Global error number \*/  void foo( void )  {  int jmpret;  jmpret = setjmp( mark ); /\* Violation \*/  }  void fphandler( int sig, int num )  {  longjmp( mark, -1 ); /\* Violation \*/  }  Repair:  Do not use the setjmp macro and the longjmp function.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |
| 2004-20\_8 | The signal handling facilities of signal.h shall not be used |
| Description: | Example:  #include <signal.h> /\* Violation \*/  Repair:  Do not use <signal.h> header.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |
| 2004-20\_9 | The input/output library stdio.h shall not be used in production code |
| Description: | Example:  #include <stdio.h> /\* Violation \*/  Repair:  Do not use <stdio.h> library.  References:  MISRA-C:2004 Guidelines for the use of the C language in critical systems  Chapter 6, Section 20 |