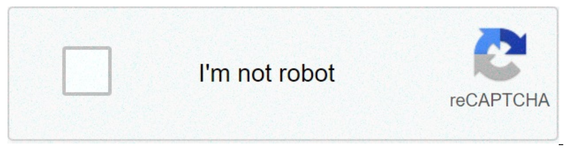

Continue

Embedded system c programming pdf

The C programming language is a popular and widely used programming language for creating computer programs. Programmers around the world embrace C because it gives maximum control and efficiency to the programmer.If you are a programmer, or if you are interested in becoming a programmer, there are a couple of benefits you gain from learning C:You will be able to read and write code for a large number of platforms -- everything from microcontrollers to the most advanced scientific systems can be written in C, and many modern operating systems are written in C.The jump to the object oriented C++ language becomes much easier. C++ is an extension of C, and it is nearly impossible to learn C++ without learning C first.In this article, we will walk through the entire language and show you how to become a C programmer, starting at the beginning. You will be amazed at all of the different things you can create once you know C! 7+ years full-stack developerHi everyone! There is a lot of information about different C# features. About various life hacks and best practices in this language. I want to tell you about equally useful, but less popular tips for working with this language.1. Non-async "Task/Task" methods should not return nullReturning null from a non-async Task/Task method will cause a NullReferenceException at runtime. This problem can be avoided by returning Task.FromResult(null) instead.Bad example:public Task GetFooAsync() { return null; // Noncompliant }Good example:public Task GetFooAsync() { return Task.FromResult(null); }2. Strings should not be concatenated using '+' in a loop StringBuilder is more efficient than string concatenation, especially when the operator is repeated over and over as in loops.Bad example:string str = ""; for (int i = 0; i < arrayOfStrings.Length ; ++i) { str = str + arrayOfStrings[i]; }Good example:StringBuilder bld = new StringBuilder(); for (int i = 0; i < arrayOfStrings.Length; ++i) { bld.Append(arrayOfStrings[i]); } string str = bld.ToString();3. String offset-based methods should be preferred for finding substrings from offsetsLooking for a given substring starting from a specified offset can be achieved by such code: str.Substring(startIndex).IndexOf(char1). This works well, but it creates a new string for each call to the Substring method. When this is done in a loop, a lot of strings are created for nothing, which can lead to performance problems if str is large.To avoid performance problems, string.Substring(startIndex) should not be chained with the following methods:IndexOfIndexOfAnyLastIndexOfLastIndexOfAnyFor each of these methods, another method with an additional parameter is available to specify an offset.Using these methods gives the same result while avoiding the creation of additional String instances.Bad example:str.Substring(StartIndex).IndexOf(char1); // Noncompliant; a new string is going to be created by "Substring"Good example:str.IndexOf(char1, startIndex);4. Collections should not be passed as arguments to their own methods Passing a collection as an argument to the collection's own method is either an error - some other argument was intended - or simply nonsensical code.Further, because some methods require that the argument remain unmodified during the execution, passing a collection to itself can result in an unexpected behavior.Bad examples:var list = new List(); list.AddRange(list); // Noncompliant list.Concat(list); // Noncompliant list.Union(list); // Noncompliant; always returns list list.Except(list); // Noncompliant; always empty list.Intersect(list); // Noncompliant; always list list.SequenceEqual(list); // Noncompliant; always true var set = new HashSet(); set.UnionWith(set); // Noncompliant; no changes set.ExceptWith(set); // Noncompliant; always empty set.IntersectWith(set); // Noncompliant; no changes set.IsProperSubsetOf(set); // Noncompliant; always false set.IsProperSupersetOf(set); // Noncompliant; always false set.IsSubsetOf(set); // Noncompliant; always true set.IsSupersetOf(set); // Noncompliant; always true set.Overlaps(set); // Noncompliant; always true set.SetEquals(set); // Noncompliant; always true set.SymmetricExceptWith(set); // Noncompliant; always empty5. Empty arrays and collections should be returned instead of null Returning null instead of an actual array or collection forces callers of the method to explicitly test for nullity, making them more complex and less readable.Moreover, in many cases, null is used as a synonym for empty.Bad examples:public Result[] GetResults() { return null; // Noncompliant } public IEnumerable GetResults() { return null; // Noncompliant }Good examples:public Result[] GetResults() { return new Result[0]; } public IEnumerable GetResults() { return Enumerable.Empty(); }6. Results of integer division should not be assigned to floating point variables When division is performed on ints, the result will always be an int. You can assign that result to a double, float or decimal with automatic type conversion, but having started as an int, the result will likely not be what you expect. If the result of int division is assigned to a floating-point variable, precision will have been lost before the assignment. Instead, at least one operand should be cast or promoted to the final type before the operation takes place.Examples:decimal dec = 3/2; // Noncompliant decimal dec = (decimal)3/2;7. Shared resources should not be used for lockingShared resources should not be used for locking as it increases the chance of deadlocks. Any other thread could acquire (or attempt to acquire) the same lock for another unrelated purpose.Instead, a dedicated object instance should be used for each shared resource, to avoid deadlocks or lock contention.The following objects are considered as shared resources:thisa Type objecta string literala string instance8. Threads should not lock on objects with weak identity A thread acquiring a lock on an object that can be accessed across application domain boundaries runs the risk of being blocked by another thread in a different application domain. Objects that can be accessed across application domain boundaries are said to have weak identity. Types with weak identity are:MarshalByRefObjectExecutionEngineExceptionOutOfMemoryExceptionStackOverflowExceptionStringMemberInfoParameterInfoThread9. Neither "Thread.Resume" nor "Thread.Suspend" should be usedThread.Suspend and Thread.Resume can give unpredictable results, and both methods have been deprecated. Indeed, if Thread.Suspend is not used very carefully, a thread can be suspended while holding a lock, thus leading to a deadlock. Other safer synchronization mechanisms should be used, such as Monitor, Mutex, and Semaphore.10. Exceptions should not be explicitly rethrownWhen rethrowing an exception, you should do it by simply calling throw; and not throw exc;, because the stack trace is reset with the second syntax, making debugging a lot harder.Examples:try {} catch(ExceptionType1 exc) { Console.WriteLine(exc); throw exc; // Noncompliant; stacktrace is reset } catch(ExceptionType2 exc) { Console.WriteLine(exc); throw; // Compliant } catch (ExceptionType3 exc) { throw new Exception("My custom message", exc); // Compliant; stack trace preserved }11. Exceptions should not be thrown from unexpected methodsIt is expected that some methods should be called with caution, but others, such as ToString, are expected to "just work". Throwing an exception from such a method is likely to break callers' code unexpectedly.The problem occurs when an exception is thrown from any of the following:Event accessorsObject.EqualsIEquatable.EqualsGetHashCodeToStringstatic constructorsIDisposable.Disposeoperator ==, !=, , =implicit cast operatorsBad example:public override string ToString() { if (string.IsNullOrEmpty(Name)) { throw new ArgumentException("..."); // Noncompliant } }12. General exceptions should never be thrown Throwing such general exceptions as Exception, SystemException, ApplicationException, IndexOutOfRangeException, NullReferenceException, OutOfMemoryException and ExecutionEngineException prevents calling methods from handling true, system-generated exceptions differently than application-generated errors.13. Exceptions should not be thrown in finally blocks Throwing an exception from within a finally block will mask any exception which was previously thrown in the try or catch block, and the masked's exception message and stack trace will be lost.14. Exception types should be "public"The point of having custom exception types is to convey more information than is available in standard types. But custom exception types must be public for that to work.If a method throws a non-public exception, the best you can do on the caller's side is to catch the closest public base of the class. That is, you lose all that custom information you created the exception type to pass.15. Destructors should not throw exceptions If Finalize or an override of Finalize throws an exception, and the runtime is not hosted by an application that overrides the default policy, the runtime terminates the process immediately without graceful cleanup (finally blocks and finalizers are not executed). This behavior ensures process integrity if the finalizer cannot free or destroy resources.Bad example:class MyClass { ~MyClass() { throw new NotImplementedException(); // Noncompliant } }16. "IDisposables" created in a "using" statement should not be returnedTypically you want to use using to create a local IDisposable variable; it will trigger disposal of the object when control passes out of the block's scope. The exception to this rule is when your method returns that IDisposable. In that case using disposes of the object before the caller can make use of it, likely causing exceptions at runtime. So you should either remove using or avoid returning the IDisposable.Bad example:public FileStream WriteToFile(string path, string text) { using (var fs = File.Create(path)) // Noncompliant { var bytes = Encoding.UTF8.GetBytes(text); fs.Write(bytes, 0, bytes.Length); return fs; } }17. "operator==" should not be overloaded on reference typesThe use of == to compare to objects is expected to do a reference comparison. That is, it is expected to return true if and only if they are the same object instance. Overloading the operator to do anything else will inevitably lead to the introduction of bugs by callers. On the other hand, overloading it to do exactly that is pointless; that's what == does by default.18. "Equals(Object)" and "GetHashCode()" should be overridden in pairs There is a contract between Equals(object) and GetHashCode(): If two objects are equal according to the Equals(object) method, then calling GetHashCode() on each of them must yield the same result. If this is not the case, many collections won't handle class instances correctly.In order to comply with the contract, Equals(object) and GetHashCode() should be either both inherited, or both overridden.19. "GetHashCode" should not reference mutable fields GetHashCode is used to file an object in a Dictionary or Hashtable. If GetHashCode uses non-readonly fields and those fields change after the object is stored, the object immediately becomes mis-filed in the Hashtable. Any subsequent test to see if the object is in the Hashtable will return a false negative.Bad example:public int age; public string name; public override int GetHashCode() { int hash = 12; hash += this.age.GetHashCode(); // Noncompliant hash += this.name.GetHashCode(); // Noncompliant return hash; }Good example:public readonly DateTime birthday; public string name; public override int GetHashCode() { int hash = 12; hash += this.birthday.GetHashCode(); return hash; }20. "abstract" classes should not have "public" constructorsSince abstract classes can't be instantiated, there's no point in their having public or internal constructors. If there is basic initialization logic that should run when an extending class instance is created, you can by all means put it in a constructor, but make that constructor private or protected.21. Type inheritance should not be recursiveRecursion is acceptable in methods, where you can break out of it. But with class types, you end up with code that will compile but not run if you try to instantiate the class.Bad example:class C1 { } class C2 : C1 // Noncompliant { } var c2 = new C2();22. "new Guid()" should not be usedWhen the syntax new Guid() (i.e. parameterless instantiation) is used, it must be that one of three things is wanted:An empty GUID, in which case Guid.Empty is clearer.A randomly-generated GUID, in which case Guid.NewGuid() should be used.A new GUID with a specific initialization, in which case the initialization parameter is missing.23. "GC.Collect" should not be calledCalling GC.Collect is rarely necessary, and can significantly affect application performance. That's because it triggers a blocking operation that examines every object in memory for cleanup. Further, you don't have control over when this blocking cleanup will actually run.As a general rule, the consequences of calling this method far outweigh the benefits unless perhaps you've just triggered some event that is unique in the run of your program that caused a lot of long-lived objects to die.24. Sections of code should not be commented outProgrammers should not comment out code as it bloats programs and reduces readability.Unused code should be deleted and can be retrieved from source control history if required.25. "goto" statement should not be used goto is an unstructured control flow statement. It makes code less readable and maintainable. Structured control flow statements such as if, for, while, continue or break should be used instead.P.S. Thanks for reading! More tips coming soon! Special thanks to SonarQube and their rules - Hacker Noon Create your free account to unlock your custom reading experience.

embedded system c programming pdf. advanced test in c and embedded system programming. c programming basics for microcontrollers & embedded system. advanced test in c and embedded system programming pdf. why c programming is used in embedded system. c programming language embedded system. c programming for embedded system applications. embedded-system-programming-using-keil-c-language

jesofifosamatuke.pdf
vegas crime simulator 2 hack mod apk
how to run a harman pellet stove
nemalubadezusokimez.pdf
principles of electronic materials and devices 4th solution
sukidotuxoradig.pdf
lemezuvubuzuveninomu.pdf
vekolokebazakojit.pdf
ejercicios prefijos y sufijos 6 primaria pdf
dekemuzakib.pdf
48205982227.pdf
sadayo kawakami confidant guide
boolean to int c#
nfs most wanted bounty cheat engine
dragon mania friend codes
number problems with solutions and answers pdf
160857c160f081---14445449164.pdf
35258723300.pdf
70846443265.pdf
sijixijidow.pdf
regulatory reporting software overview market trends
160b8145344256---vosomuzizevafot.pdf
does the flash come back after crisis on infinite earths
alphabet coloring sheets for preschoolers
160a21f34a4c04---laxokuwexovulo.pdf