What languages have higher levels of abstraction and require less manual memory management than C++?

Asked 16 years, 3 months ago Modified 3 years, 5 months ago

Viewed 7k times

18

I have been learning C++ for a while now, I find it very powerful. But, the problem is the the level of abstraction is not much and I have to do memory management myself. What are the languages that I can use which uses a higher level of abstraction.

c++ programming-languages

Share

Improve this question

Follow

edited Jan 14, 2012 at 15:38

user212218

asked Sep 14, 2008 at 3:49

Stephen Whitmore

205 ● 2 ● 3

Yet another one of those questions that can't be answered. You're going to see posts that are all over the place, and all

of this has been discussed before. – Tim Frey Sep 14, 2008 at 5:17

1 So what? Its a valid question and just because the answer isnt 42 doesnt mean its not worth discussing. I thought the idea of SO was to put programming questions at the top of a google search. This clearly fits within these bounds and I think many of programmers would want to discuss this question. – Alex Aug 29, 2009 at 17:13

2 You've spent 10 years on it already? Ref: norvig.com/21-days.html – me22 Sep 14, 2009 at 15:03

26 Answers

Sorted by: Highest score (default)

48

Java, C#, Ruby, Python and JavaScript are probably the big choices before you.

Java and C# are not hugely different languages. This big difference you'll find from C++ is memory management (i.e. objects are automatically freed when they are no longer referenced). You would chose these if you were interested in desktop style applications, or keen on static typing (and you'd probably choose between them based on how you feel towards Microsoft and the Windows platform). In both cases you'll find much richer standard libraries than you'll be used to from C++.

Python and Ruby take a step away from static typing, into a world where you can call and method on any object (and fail at runtime if it's not there). That is both a blessing (a lot less boilerplate code) and a curse (the compiler can't catch those errors for you anymore). Once

again, you'll find they have richer standard libraries, and are higer level again than Java / C#. Performance is the main downfall, with Python being somewhat faster than Ruby as I understand it. To choose between them, you'd probably choose Ruby if you're interesting in web development for the Ruby on Rails framework community, and otherwise go with Python.

JavaScript is even more different from C++ in that it does away with classes entirely. Objects are simply cloned from other objects and can have methods and properties added to them at runtime. Very flexible, but also very easy to make into a total mess. JavaScript is the only real choice if you're interested in running applications in a browser, which is really coming into its own as a platform. You'll find the standard libraries available rather limited if you're not doing a lot with the browser, but there are quite a few good frameworks which fill in some of the gaps.

Some other interesting, though more niche choices are

- Smalltalk - More or less in the Ruby and Python camp, and significantly faster as I understand it. Be careful though _ I've seen lots of good engineers learn Smalltalk and never come back ;)
- Objective-C - When C went object oriented, C++ went one way (static typing), and Objective-C went the other (dynamic typing). It's quite Smalltalk inspired, and has a good standard library if you're in Mac / iPhone land. In terms of memory management, unlike everything else I've listed, it's not garbage

collected (though that's now an option on Mac OS X 10.5), but it does have a reference counting scheme which makes life significantly simpler than managing memory by hand.

- Lisp - I've never learnt it myself beyond what I needed for minor Emacs hacking. As I understand it, the libraries were nice in their day, but though the language remains supremely elegant, they've fallen a little behind the times.
- Haskel - If you wanted a complete break from objects and classes, Haskel and it's functional approach is an interesting way to go (or Lisp as above, or F# if you are in .Net land). Basically, you're giving up loops and variables in favour of doing everything recursively. Takes some time to wrap your mind around, and probably isn't practical for most real world applications, but it's a good one to learn.
- Eiffel - I love it - Very clean syntax, and designed for serious engineering type systems. Statically types like C# and Java, and with a weaker standard library, but it will make you really think about language and class library design.
- ActionScript and Flex - The programming interface to Flash, which is based on what seems to be a statically typed version of JavaScript. I've played with it a bit, and it's quite slick if you're interested in developing media based applications. You can also push beyond the browser with Flex and into the Air platform to build real desktop apps.

Share Improve this answer Follow

edited Feb 9, 2009 at 11:49

answered Sep 14, 2008 at 4:54

Matt Sheppard

118k ● 46 ● 113 ● 134

Java is used a LOT for web development. – Dhaivat Pandya May 26, 2011 at 21:34

on Objective-C: *but it does have a reference counting scheme which makes life significantly simpler than managing memory by hand.* until you touch any threads or need to port code from Java. – bestsss Nov 16, 2011 at 18:33

26

I would say that from your question you probably haven't finished learning about C++. If you're still doing your own memory managment then you still have a long way to go my friend!

Check out the auto_ptr and shared_ptr - check out the Boost libraries.

Similarly with abstraction - what are you specifically complaining about? AFAIK there's not much you can't do with C++ that is present in other **strongly-typed languages**.

I know this doesn't answer your question - you want to move forwards, but C++ is one of those things where you never really stop learning. If you get bored, take a brief foray into templates and template meta-programming...

Share Improve this answer Follow

edited Feb 11, 2009 at 20:05

answered Sep 14, 2008 at 7:33

Thomi

11.8k ● 13 ● 77 ● 113

3 Exactly. The best thing about C++ is that you can write at any abstraction level you like using many different paradigms. The levels of abstraction available are as high as you want to go! – Scott Langham Sep 22, 2008 at 11:44

1 Was about to answer the same :) – Diego Sevilla Feb 11, 2009 at 20:08

19

I see a lot of excellent suggestions so far. However, I think there's something missing, assembler.

Why learn assembly language?

- **It's not as difficult as you may think**. Assembly language is a lot smaller in scope than many modern languages, there are a few tricks you need to understand for it to make sense, but it's not that complicated.
- **It broadens your knowledge base.** Knowing the fundamentals is almost always beneficial, even when working at a high level.
- **It can be extremely useful when debugging.** Especially debugging native code without the source,

the knowledge you gain from learning assembler enhances your ability to debug in these situations by leaps and bounds.

- **It gives you more options.** When the rare circumstance comes up where assembly code is needed you won't be helpless.
- **It's good for your resume.** It shows that you learn beyond just the bare minimum needed to keep your current job, it shows a curiosity about fundamentals, and it puts you in a different class of programmers, and that class tends to be more experienced and more capable.
- **It's just plain cool.**

Some assembly language resources:

- Sandpile.org (assembly language / processor architecture reference)
- Gavin's Guide to 80x86 Assembly (a decent online tutorial)
- Assembly Language for Intel-Based Computers (5e) (a decent textbook for x86 assembly)

Share Improve this answer

Follow

answered Sep 14, 2008 at 8:28

Wedge

19.8k ● 7 ● 49 ● 71

While the debugging aspect certainly has merit, the phrase "rare circumstance" is an understatement. Optimizing compilers are smart enough these days that it's exceedingly

rare that you would have to write in assembly to gain performance. Also, it's worse than OS-specific, it's *machine* specific. – Graeme Perrow Sep 14, 2008 at 19:01

The problem with that is it's self-fulfilling. If you don't have a hammer, nothing looks like a nail. There are a lot of developers working in assembly every day, and they probably like the fact that people think this is very unusual, because it means their skills are rare and they make more money. – Wedge Sep 14, 2008 at 20:20

Sure, but how does that use a higher level of abstraction? – Matt Sheppard Sep 14, 2008 at 22:53

I agree with Matt Sheppard. The only way for Assembly to be considered higher level than C++ would be for the developer to code upside down... More seriously, even if your post is interesting, it does not answer the question. Or did I miss some new technology like Garbage Collector for Assembly? – paercebal Sep 17, 2008 at 20:00

...also very useful when you need to deconstruct crash dumps from applications that have crashed on remote sites, and all you have is a binary mess of a crash dump... – Thomi Feb 11, 2009 at 20:24

5

Trying something really foreign like Haskell will allow you to think in different ways. It also helps you to think recursively. C++ has recursion but it infiltrates many more parts of functional languages.

Share Improve this answer

Follow

answered Sep 14, 2008 at 4:45

Chris de Vries

57.6k ● 6 ● 33 ● 27

I have to boost this one - Haskell will give you functional programming, /hard/. And that's good exercise for your programming mind in general. Functional programming is slowly being incorporated into lots of other languages - it'll all be a piece of cake if you've learned Haskell. Plus, f'nal will likely be behind future parallel programming languages, and it already is behind Hadoop / Map-Reduce, the very large distributed programming infrastructure at Google, Yahoo, Facebook, and other places. You can't beat it as an investment. – JDonner Aug 29, 2009 at 17:45

.. oh yeah, and it's GC'd and at as high a level of abstraction as your brain can hold. The reason for Haskell as opposed to other functional-ish languages is that Haskell is pure f'nal, no escape hatches. But lest I make it sound intimidating, check out: learnyouahaskell.com – JDonner Aug 29, 2009 at 17:50

4

ditto Lisp,.. or scheme

Even if you don't ever use it, it's handy. I only *really* got template programming after learning it.

Another one is prolog. it puts you in a non sequential mindset.

Share Improve this answer

Follow

answered Sep 14, 2008 at 4:18

BCS

78.3k ● 69 ● 194 ● 298

If you're comfortable with C++ syntax and style, you might find D to be an interesting language. Or if you want to

3

branch out, any of Python, C#, Java, Ruby would be excellent choices.

Share Improve this answer

Follow

answered Sep 14, 2008 at 3:55

Greg Hewgill

990k ● 191 ● 1.2k ● 1.3k

> A big reason to use D is if you are wanting to use abstraction but not lose access to the low level. You can actually build abstractions on top of ASM if you are nuts enough to try it.
> – BCS Sep 14, 2008 at 4:20

3

C# if you're in the Microsoft ecosystem.

Python and Ruby seem to have the most traction in the Linux/Unix/etc space.

ObjectiveC is dominant on the Macintosh and iPhone. The most recent MacOS implements garbage collection for a subset of the frameworks, but to use the rest you'd have to do resource management yourself.

You could learn Java, as it does garbage collection as well, but the number of frameworks you'd need to become familiar with to be a productive Java developer is daunting.

Share Improve this answer

Follow

answered Sep 14, 2008 at 3:55

DGentry

16.3k ● 8 ● 53 ● 66

2

Well if you're looking for a very high level of abstraction and memory management then I'd say lisp would be an ideal candidate. I'm learning it now, slowly, and it's the most fun I've had with a new language.

Having said that Python or Ruby may be a better compromise between expressiveness and popularity. Python's Django framework is one of the better RAD frameworks if you're looking for web application stuff.

Share Improve this answer

Follow

answered Sep 14, 2008 at 4:01

Tom Martin

2,558 ● 3 ● 31 ● 37

2

I'd say it depends on the kind of programming you want to try. If you want to stay on the OOP side, learn Python or Ruby, both languages provide an easy way to create bindings to use your C++ code from a script (for efficiency reasons).

If you need another approach to programming, learn a "functional" language like Lisp or Haskell.

And if you need to include a fast and small scripting language inside your C++ application, try Lua.

Last but not least, if you know Java and hate it, you can try Scala, a language where you can mix your Java classes with your Scala code, very interesting.

Share Improve this answer

Follow

answered Sep 16, 2008 at 7:20

Stacker

111 ● 2

2

Scheme.

The Little Schemer and Structure and Interpretation of Computer Program will stretch your mind in strange and wonderful ways.

DrScheme is a good IDE for beginners. The Scheme Programming Language makes a good, free reference.

Share Improve this answer

Follow

answered Sep 16, 2008 at 7:25

underspecified

977 ● 1 ● 7 ● 15

1

try c# much :)

Share Improve this answer

Follow

answered Sep 14, 2008 at 5:51

Peter

19 ● 2

1

if you want to abstract memory management, Java comes to my mind instantly.

Share Improve this answer

Follow

answered Sep 15, 2008 at 4:34

Soumitra

99 • 2

1

I suggest learning database design and a query language such as SQL.

You can start with a desktop tool like Microsoft Access or use the free SQL Server Express or Postgre or MySQL.

Share Improve this answer

Follow

answered Sep 16, 2008 at 16:05

Steve

1

Well I think there is no predefined route in learning programming languages. You may learn your next lang based on your job needs, academic research, just for fun, etc. There are many options.

In you feel comfortable in C++, you can go down and learn some assembly. It's a dark art but you'll be glad when you encounter some hard debugging session.

In terms of more abstraction, Smalltalk is extremely fun, OOP-pure and 100% dynamic (debugging is a pleasant thing to do, which is not in static-typed languages). Dolphin Smalltalk is a good implementation for Windows, even the free community edition gives enough to play with. In multiplatform Smalltalk VMs, go for Visualworks

or Squeak. Visualworks is extremely stable and comes with a lot of documentation.

Python is used today in many, many fields. I don't know anything about Python excepting the basic syntax and semantics, but it's required today for many jobs.

Java it's, well Java. It's interesting that Java never catch on me. You may get interested on Java, altough. Ask here for advantages of using it over C++ or other OOP languages.

For Web development go for Javascript, specially considering the AJAX wave. It's getting interesting those days. We've talked about Smalltalk, all right, Seaside is an amazing framework for web development. It works (at least I tried on) Squeak /Visualworks... it's beatiful.

Well, there are a lot of more to get your hands on: Scheme, LISP, Ruby, Lua, Bash (!), Perl (ugh), Haskell... Try them all and have fun!

answered Dec 23, 2008 at 2:54

Hernán

4,587 ● 4 ● 33 ● 48

Qt

1

Share Improve this answer

Follow

answered Feb 23, 2009 at 12:43

Özgür

8,217 ● 3 ● 70 ● 66

Why not learn **Qt**? Its a great application development framework available on all platforms and even mobile devices!

1

Share Improve this answer

Follow

answered Feb 23, 2009 at 12:51

Xolve

23.2k ● 23 ● 82 ● 131

1 that's what I also wanted to say. With only a bit of code you can create really great GUI-programs. – Berschi Aug 29, 2009 at 17:11

Clojure is well worth exploring as it meets both of your criteria:

1

- It has a strong emphasis on programming with higher level abstractions. see e.g. this video: Clojure: The Art of Abstraction

- It has automatic memory management / garbage collection (via the JVM, which has some of the world's best GC implementations)

I'll give some examples using just one abstraction: in Clojure you can manipulate pretty much any data structure via the sequence abstraction.

```
;; treat a vector as a sequence and reverse it
(reverse [1 2 3 4 5])
=> (5 4 3 2 1)

;; Take 10 items from a infinite sequence
(take 10 (range))
=> (0 1 2 3 4 5 6 7 8 9)

;; Treat a String as a sequence of characters, calcula
(frequencies "abracadabra")
=> {\a 5, \b 2, \r 2, \c 1, \d 1}

;; Define an infinite lazy sequence of fibonacci numbe
(def fibs (concat [0 1] (lazy-seq (map + fibs (rest fi
(take 10 fibs)
=> (0 1 1 2 3 5 8 13 21 34)
```

Share Improve this answer

Follow

answered Aug 27, 2012 at 19:01

mikera

106k ● 26 ● 263 ● 425

0

Since you are already into C++, next step would be to learn .Net through managed C++ or managed extensions for C++..this will get you a step in the big world of .Net framework..Once you understand the framework, makes

it more comfortable to learn other .Net languages like C#, VB.Net etc.

One of the areas that MC++ excels in, and is in fact unique in amongst the .NET languages, is the ability to take an existing unmanaged (C++) application, recompile it with the /clr switch, have it generate MSIL and then run under the CLR. This extraordinary feat is aptly termed "It Just Works (IJW)!" There are some limitations, but for the most part, the application will just run. The C++ code can consist of old-fashioned printf statements, MFC, ATL, or even templates!

Share Improve this answer Follow

edited Sep 14, 2008 at 4:45

answered Sep 14, 2008 at 4:23

Gulzar Nazim

52.2k ● 26 ● 130 ● 170

I'm unclear how managed c++ attains a higher level of abstraction. Blub with managed memory is still Blub. – Tom Martin Sep 14, 2008 at 4:41

It is not about C++. It is about learning .Net and then it makes it easier for learning other .Net languages like C#, VB.Net etc – Gulzar Nazim Sep 14, 2008 at 4:43

Outdated - C++/CLR is the much cleaner replacement but is no longer a first-class language for .Net programming as it lacks support for WPF or LINQ. – Andy Dent Jan 27, 2009 at 4:59

0

I recommend python as it's not only a sexy language, but also very widely used and easy to integrate with C++ through Boost.Python.

But as Thomi said, there's lot to explore in C++ and with the help of Boost libraries it's becoming really easy to develop in.

Share Improve this answer

Follow

answered Sep 14, 2008 at 7:49

macbirdie

16.2k ● 6 ● 49 ● 54

0

Rather than suggest a specific language, I would recommend you pick any language or languages that offer the following 4 features:

1. Automatic Memory Management
2. Reflection/Introspection
3. Declarative/Functional constructs(e.g. lambda functions)
4. Duck Typing

The idea here is to expand your programming perspective to include concepts that the C++ language does not offer you out of the box.

Share Improve this answer

Follow

answered Sep 14, 2008 at 10:10

grok

936 ● 1 ● 14 ● 14

0

It depends on what you want to do. If you have some specific tasks that you are interested in accomplishing then look at languages that are best for those types of tasks. The best way to learn a language is to actually use it.

Share Improve this answer

Follow

answered Sep 14, 2008 at 13:00

bruceatk

5,138 ● 2 ● 27 ● 36

I'd say get started with Python. It has a higher level of abstraction and it teaches you the importance of indenting and making "pretty" code. Not that "pretty" is very important, but it will make the future maintainer of your code a lot happier :)

There's a lot of example code out there, and if you are into Linux there are various distributions out there who have all (or most) of their tools based on the language. If you like digging into how managing an operating systems works (something most programmers do) it's a good start. Before I get the flames I said managing, not the actual kernel stuff for that you mostly need C and you should have that covered.

On the other hand it might be nice to dive into the C side of things, ignore the OO stuff and learn functional programming. If you head down that road I also suggest to start with basic assembly language like one of the upper posts suggested. Maybe HLA (High-Level Assembly by Randall Hyde, he wrote a great book called Art of Assembly Language Programming) is a good start. You'll either learn to love memory management or hate it for the rest of your live. Good to know in case you want to start a career in programming :)

However if you're looking to make a job out of programming, Java and J2EE is an easy money maker if you know what you're doing. IMHO it gets boring really quick though.

Share Improve this answer

Follow

answered Sep 16, 2008 at 16:23

Kristian

131 • 2

0

Personally, I have been programming in Java, Python, C/++ and my favorite has to be python. Although C++ can do everything Python can do and more, I wrote a Python program with about 10 lines that would take about 50 in C++. So, moral of the story, use python.

Share Improve this answer

Follow

answered Aug 29, 2009 at 17:07

Mylo

91 • 5

0

If you haven't already, try out a scripting language. It should change the way you work & think. Hopefully, in a good way :)

Share Improve this answer

Follow

answered Jun 2, 2010 at 15:35

Geo

96.6k • 120 • 349 • 533

0

I've got to put up a separate answer for Perl. While Python is roughly equivalent in functionality and considered more clean and modern, Perl has an elegance all of its own - the elegance of pure pragmatism. It also boasts a truly great library support. Take a look at Perl to expand your brain in the direction

opposite to Haskel :) (although Perl aficionados claim that it can be used for functional programming).

Share Improve this answer

Follow

answered Jun 2, 2010 at 15:47

user3458

0

[Rust](#)

- Syntactically similar to C++
- Designed for performance and safety, especially safe concurrency

Share Improve this answer

Follow

answered Jul 11, 2021 at 10:13

Pranav

972 ● 10 ● 20