Object-Oriented Programming in C

Anne Gatchell

16 November 2012

Points to Cover

- GTK+ - http://en.wikipedia.org/wiki/GTK%2B
- GObject http://en.wikipedia.org/wiki/GObject
- OOC Book
- why would you do this
- GObject ref manual http://developer.gnome.org/gobject/stable/pr01.html
- disadvantages: optimization, look at x86 code
- C vs C++: arguments and compromise

Points to Explore

- Can we treat ANSI-C as an object-oriented language?
- How could we do this?
- Why would we do this, instead of using C++?
  - C/C++ wars
- Examples of OOC
- GTK+ and GObject
- Conclusions

Object-Oriented Programming

- A departure from Functional programming, C's specialty
- First things first: A basic definition of object-oriented programming:
  - *Programming with a focus on entities that have data and associated responsibilities*
- In C++, we use a Class to define a template for objects
  - The class contains information on what data the object instance should contain and what tasks that object will perform
- In C, we don't have Classes

Can we treat ANSI-C as an OO Language?

- In C, we have some data types: int, char, double, etc.
- These data types are certain types of values
- They are intimately tied to their implementation (eg. char has 256 possible values, double is hardly a mathematical real number)
- But, we can also view data types as having the same definition of Objects:
  - "A set of values plus operations to work with them" - Schreiner

Can we treat ANSI-C as an OO Language?

- So, we basically want to be able to create Abstract Data Types:
  - They will be able to conceal their implementation details from the user,
  - which will aid the user in dividing and conquering their code to make it more modular
- How can we implement this?
- With structs and void * pointers
- Basic Set implementation example from Axel-Tobias Schreiner's Object-Oriented Programming in ANSI-C book
  - To take a look at how we can achieve true abstraction in C

Set (1)

Let us implement a Set of elements with methods for add, find and drop. Each method takes a set and an element, and returns the element added to, found in, or removed from the list.

Notice the use of generic void * pointers. This means that the user of these methods cannot possibly glean anything about the implementation of Set

Set.h:

```
#ifndef SET_H #define SET_H

extern const void * Set;

void * add (void * set, const void * element);
void * find (const void * set, const void * element); void * drop (void * set, const void * element);
int contains (const void * set, const void * element);

#endif
```

Set (2)

We are using pointers to refer to sets, rather than Typedefs, so we need a way to obtain a set and delete a set. These methods are declared in new.h.

new.h:

```
void * new (const void * type, ...);
void delete (void * item);
```

The new function accepts a type (ie. Set) and zero or more arguments for initialization and returns a pointer to that abstract data type.

Now, we need an Object type, so that we have something to put in the Set!

Object.h:

```
extern const void * Object; /* new(Object); */
int differ (const void * a, const void * b);
```

Set(3)

Example application:

This application should output "ok"

```
#include <stdio.h>
#include "new.h" #include "Object.h" #include "Set.h"
int main ()
{
void * s = new(Set);
void * a = add(s, new(Object)); void * b = add(s, new(Object)); void * c = new(Object);
if (contains(s, a) && contains(s, b))
     puts("ok");
if (contains(s, c))
     puts("contains?");
if (differ(a, add(s, a)))
     puts("differ?");
if (contains(s, drop(s, a)))
     puts("drop?");
delete(drop(s, b)); delete(drop(s, c));
return 0;
}
```

Set(4)

The implementation for this small program assumes that each object stores no information and belongs to at most one set. We represent objects and sets as integers that are indexes to a heap[] array. If an object is a member of the set, then its array element contains the number of the set.

Since this example's purpose is to demonstrate the abstraction of the methods, we do not need to have the objects hold data right now.

Set.c.

```
#if ! defined MANY || MANY < 1 #define MANY 10
#endif
static int heap [MANY];
void * new (const void * type, ...)
{
int * p; /* & heap[1..] */
for (p = heap + 1; p < heap + MANY; ++ p) if (! * p)
break;
assert(p < heap + MANY);
* p = MANY; return p;
}
```

Set (5)

We need to make sure that the item's number is within the bounds of the heap, and then we can set it to 0.

Set.c cont.

```
void delete (void * _item)
{
int * item = _item;
if (item)
      { assert(item > heap && item < heap + MANY);
      * item = 0;
      }
}
```

Set (6)

Set.c cont.

```
void * add (void * _set, const void * _element)
{
int * set = _set;
const int * element = _element;
assert(set > heap && set < heap + MANY); //
assert(* set == MANY); //Make sure the set does not belong to another set
assert(element > heap && element < heap + MANY);
if (* element == MANY)
    * (int *) element = set – heap;
else
    assert(* element == set – heap);
return (void *) element;
}
```

Set (7)

Set.c cont.

```
void * find (const void * _set, const void * _element)
{
const int * set = _set;
const int * element = _element;
assert(set > heap && set < heap + MANY); assert(* set == MANY);
assert(element > heap && element < heap + MANY); assert(* element);
return * element == set – heap ? (void *) element : 0;
}
int contains (const void * _set, const void * _element) //Converts the result of find into a
Truth value
{
return find(_set, _element) != 0;
}
```

Set (8)

Set.c cont.

```
void * drop (void *_set, const void * _element)
 {
 int * element =find(_set, _element);
if (element)
      * element = MANY;
return element;
}

int differ (const void * a, const void * b)
{
return a != b;
}
const void * Set;
const void * Object;
```

Phew!!

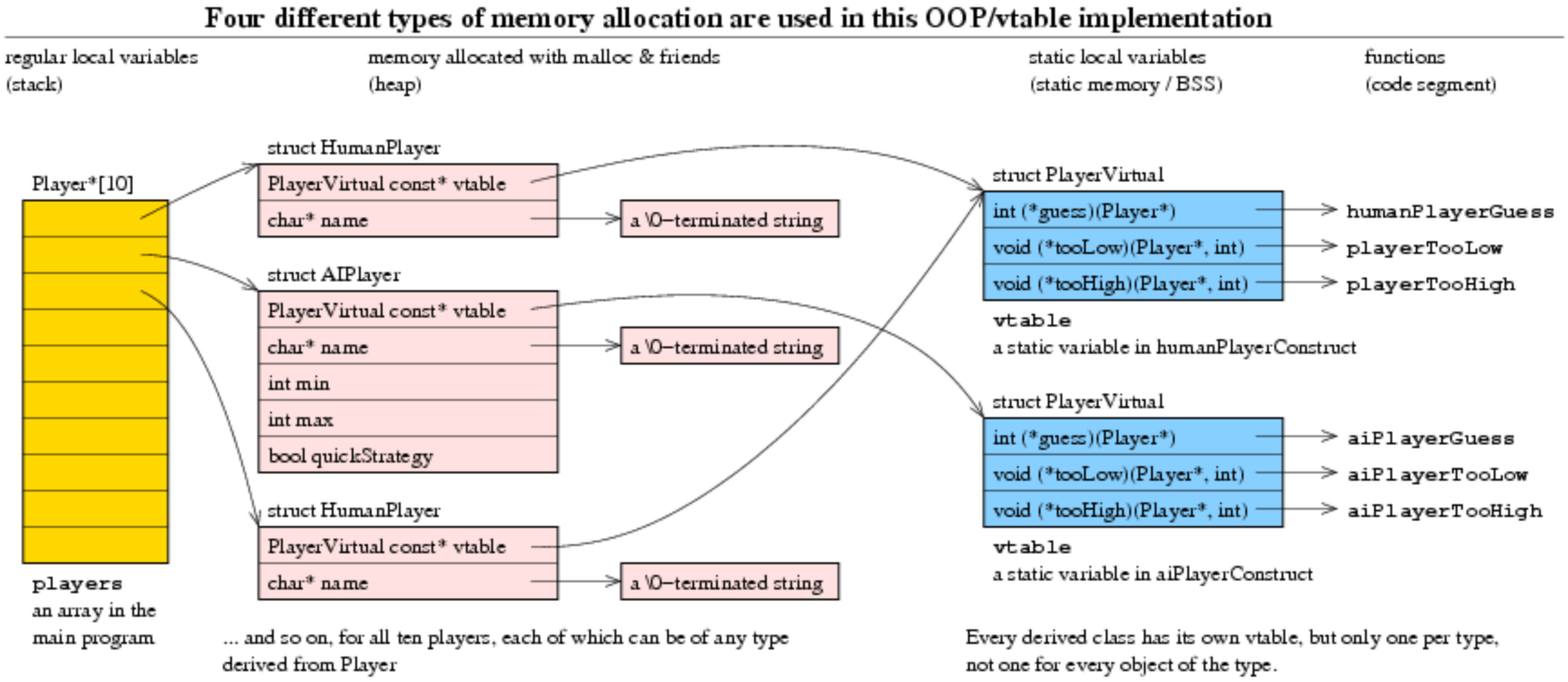
- The takeaway from all that C code for a simple set is that we have something very much like a set in Python
- We can add, find, or delete any type of object from our Set data type, without any worries about the implementation beyond the question of, "Does this behave like a mathematical set, and would a set meet my needs?"
- The application code can only see the header file, in which a descriptor pointer represents the data type, and the operations take and return generic pointers
- Usually, the descriptor pointer would point to a constant size_t to indicate the size of the data type

Other Methods

- Many have published or posted advice for people who want to try object orient programming in C
- A key factor that will determine the complexity of the implementation is whether or not the programmer wants to be able to actually keep members private and totally abstract
- Some may just want the organization of OO-Design, but decide to just hold themselves to a *contract* that says they will not deviate from the permissions that are outlined in OO principles

StackOverflow Example (1)

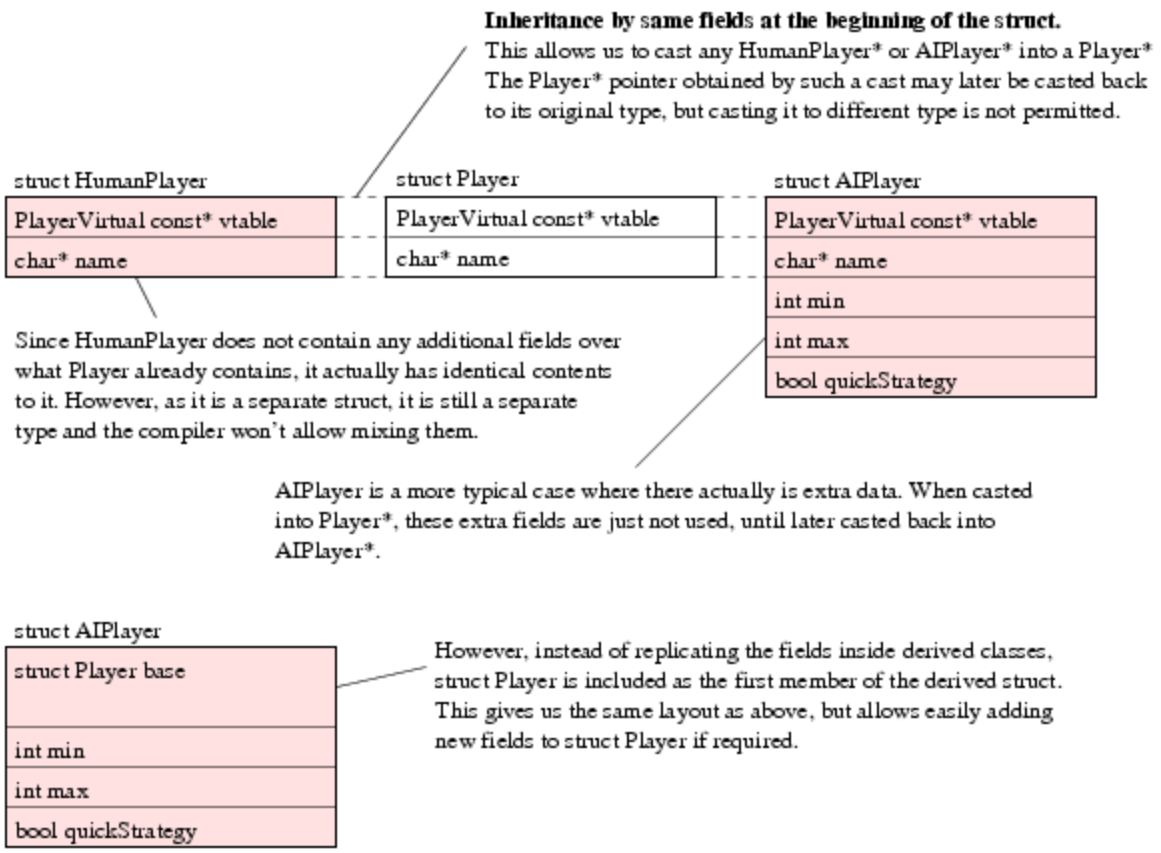
One SO member Tronic posted this diagram of implementing polymorphism with regular functions and vtables to contain the functions for a given type



StackOverflow Example (2)

Continued. The diagram is pretty descriptive, but the whole structure is quite elegant. The properties of structs lend themselves beautifully to polymorphism

HumanPlayer and AIPlayer both derive from the Player struct. The AIPlayer uses all fields from the Player struct, and also adds some. It can be cast to a Player and treated as such (only the Player properties would be exploitable) and then it can be cast back to an AIPlayer.



Embedded Systems Programmers

- Embedded Systems Programmers often need to use C because that is either the only language that their device supports/compiles, or because it would be far easier and smaller to implement a C compiler than a compiler for a higher level language
- See the following article that helps C programmers see how to write C programs that are equivalent to their C++ counterparts

Why Would We Do This?

- Object-oriented C is a common question topic online
- The reasons for using C in an Object-Oriented method can range from preference to necessity
- Embedded developers who are restricted to C many desire to use object-oriented design methodologies
- To get a picture of why people might choose OO-C over C++, we can look at some of the reasons people use C over C++
- As Axel-Tobias Schreiner states in his book, it is a great exercise to understand and appreciate how object-oriented design works
- Or, as one StackExchange member commented on the Tronic format, it is a great way to understand the inner workings of other OO languages, like Java

Why choose C over C++?

- Writing low level or embedded code
- C++ compiler is not good at a particular optimization
- Application does not lend itself to being object-oriented
- Must meet industry guidelines; easier to prove and test for in C
- C compiler is smaller and ubiquitous
- C is (often) faster
- Tools you are using are written in C (OpenGL, OpenCL, etc)
-

Why choose C over C++?

Popular applications that use C:

- Linux
- Git
- GTK+
  - object-oriented C! more on this later

C vs. C++ War

- Reading arguments about the merits of C or C++
- It gets very heated
- Interesting reading:
  - Linus Torvalds' rant about C http://thread.gmane.org/gmane.comp.version-control.git/57643/focus=57918
  - A response to said rant http://warp.povusers.org/OpenLetters/ResponseToTorvalds.html

Finding peace between C and C++

- C has many advantages
- If one needs to use C for a system that lends itself to object-oriented programming, OOC is a great way to deal with that
- Rather than be dogmatic about whether C or C++ is better, look at the strengths and weaknesses of the languages and whether a Functional Decomposition approach is adequate or an Object-Oriented approach would be better for your purposes

Practical Examples of OOC

- GTK+ (GIMP Toolkit) is a multi-platform toolkit for creating GUIs
- It is built in C, and it is object oriented
- It uses GLib(and GObject)
- You can clone the GTK code using
  - git clone git://git.gnome.org/gtk+
- It is interesting the see the very organized (and large) collection of object oriented "class" in C

GObject (GLib Object System)

- If you want to write an OOC program, it may be worth visiting <http://developer.gnome.org/gobject/stable/index.html> and checking out the GObject library
- It provides a generic type system that allows for inheritance and memory management
  - Basically all the things that Axel-Tobias Schreiner's implementation does

Conclusions

- Yes, you can use C in an object-oriented fashion
- It is worth it?
  - That's a tougher question
  - Answer will vary with project
  - Using a library someone else has written (GObject, Schreiner) makes it much easier to get the the meat of the design
- If anything, it will make you a better C programmer, and probably a better OO programmer
- Try it on for size!

References

- *Object-Oriented Programming with ANSI-C* by Axel-Tobias Schreiner
- http://stackoverflow.com/questions/2181079/object-oriented-programming-in-c?lq=1
- Stack Overflow: Why would anybody use C over C++? [closed] http://stackoverflow.com/questions/497786/why-would-anybody-use-c-over-c
- GTK+ Overview http://www.gtk.org/overview.php
- Stack Overflow: Object Oriented Programming in C http://stackoverflow.com/questions/2181079/object-oriented-programming-in-c?lq=1
- GObject Reference manual http://developer.gnome.org/gobject/stable/pr01.html
- Object Oriented Programming in C (for Embedded developers) http://www.eventhelix.com/realtimemantra/basics/object_oriented_programming_in_c.htm
- GTK+ Project http://www.gtk.org/download/index.php
- Some proof(?) that C is faster than C++ for equivalent programs http://unthought.net/c++/c_vs_c++.html