```
    double d_d;  // size =  8;  alignment = 8
    int    d_i;  // size =  4;  alignment = 4
    char   d_c;  // size =  1;  alignment = 1
};               // size = 16;  alignment = 8
```

Both **alignof**(Wasteful) and **alignof**(Optimal) are 8 on our platform, but **sizeof**(Wasteful) is 24, whereas **sizeof**(Optimal) is only 16. Even though these two **struct**s contain the very same data members, the individual alignment requirements of these members forces the compiler to insert more total padding between the data members in Wasteful than is necessary in Optimal:

```
struct Wasteful
{
    char   d_c;           // size =  1;  alignment = 1
    char   padding_0[7];  // size =  7
    double d_d;           // size =  8;  alignment = 8
    int    d_i;           // size =  4;  alignment = 4
    char   padding_1[4];  // size =  4
};                        // size = 24;  alignment = 8

struct Optimal
{
    double d_d;           // size =  8;  alignment = 8
    int    d_i;           // size =  4;  alignment = 4
    char   d_c;           // size =  1;  alignment = 1
    char   padding_0[3];  // size =  3
};                        // size = 16;  alignment = 8
```

Determining if a given buffer is sufficiently aligned

The **alignof** operator can be used to determine if a given, e.g., **char**, buffer is suitably aligned for storing an object of arbitrary type. As an example, consider the task of creating a **value-semantic** class, MyAny, that represents an object of arbitrary type[2]:

[2]The C++17 Standard Library provides the ~~nontemplate~~ class std::any, which is a type-safe container for single values of *any* ~~regular type~~. The implementation strategies surrounding alignment for std::any in both libstdc++ and libc++ closely mirror those used to implement the simplified MyAny class presented here. Note that std::any also records the current **typeid** on construction or assignment, which can be queried with the type **member function** to determine, at run time, whether a specified type is currently the active one:

```
#include <any>  // std::any
void f(const std::any& object)
{
    if (object.type() == typeid(int)) { /*...*/ }
}
```