What is the purpose of templates?

Templates share source code among structurally similar families of classes and functions.
Many data structures and algorithms can be defined independently of the type of data they manipulate. A template allows the separation of the typedependent part from the type-independent part. The result is a significant amount of code sharing.

A template is like a cookie cutter: all the cookies it creates have the same basic shape, though they might be made from different kinds of dough. A class template describes how to build classes that implement the same data structure and algorithm, and a function template describes how to build functions that implement the same algorithm.

In other languages, these facilities are sometimes called parameterized types or genericity. Prior to templates, macros were used as a means of implementing generics. But the results were so poor that templates have superceded them.

What is an alternative to dynamic type checking with containers?

Templates offer a viable alternative hen working with containers.

In the past, some container classes were designed assuming the existence of some kind of master base class. This has been called the based object approach. In particular, it was common to encounter container classes that inserted or extracted elements that were pointers to a single base class, typically called Object.

Applying the based object approach to containers makes it hard to mix two or more class libraries. For example, it may not be possible to put an object from one library into a container from another library, since the master base classes from the two libraries normally won’t match exactly.

In general, this approach can and should be avoided through the use of templates or design patterns. The particular problem of extensible container classes has been elegantly solved in the standard C++ container classes by using templates and iterators

Does the fact that an array-of Derived can be passed as an array-of Base mean that arrays are bad?

Yes, arrays are dangerous. Use template container classes instead.

Compared to a C++ array, a template container class catches more errors at compile time, thus reducing the reliance on runtime testing. For example, if the standard template vector<T> had been used, the previous attempt to pass a vector<Derived> as a vector<Base> would have been caught at compile time.

Templates allow the compiler to distinguish between a pointer to a thing and a reference to an array-of things. In contrast, when C++ arrays were used , the compiler wasn’t able to detect the error of passing an array-of Derived as if it were a kind-of array-of Base. The compiler detects this error if templates are used.