```
/*
 * File:   PodString.h
 * Author: ihersht
 *
 * Created on February 21, 2014, 11:57 AM
 */


#ifndef CPP_TOOLS_POD_STRING_H
#define CPP_TOOLS_POD_STRING_H

#include <string>
#include <cstring>
#include <cstdint>
#include <algorithm>
#include <type_traits>
#include <array>

namespace cpp_tools
{
#if(  __SIZEOF_INT128__  == 16)
        using int128_t =  __int128_t;
        using uint128_t =  __uint128_t;
#else
        using int128_t =  std::array<std::int64_t, 2>;
        using uint128_t = std::array<, 2>;
#endif
namespace PodStringImpl{

template<std::size_t MaxSize > struct ImplType;
template<> struct ImplType<1> {using type =  std::uint16_t;};
template<> struct ImplType<3> {using type =  std::uint32_t;};
template<> struct ImplType<7> {using type =  std::uint64_t;};
template<> struct ImplType<15> {using type =  uint128_t;};
template<> struct ImplType<23> {using type =  std::array<std::int64_t, 3>;};
template<> struct ImplType<31> {using type =  std::array<std::int64_t, 4>;};



template<std::size_t MaxSize>  struct PodString {

 using value_type = typename ImplType<MaxSize>::type;
 static_assert(MaxSize < sizeof(value_type), " Coding error. maxSize >= sizeof(value_type)" );
 static constexpr bool  IsArithmetic() { return (alignof(value_type) ==sizeof(value_type))  ; }
 enum{MAX_SIZE = MaxSize};


 PodString(value_type value)  noexcept  : m_value{value} {}
 PodString(const  char* value, std::size_t len) noexcept{
  if(len < sizeof(value_type)) {
                memcpy(&m_value, value, len);
        }
  }

 PodString(const  char* c_str) noexcept : PodString(c_str,  strlen(c_str)  ) { }
 PodString(const  std::string &str) noexcept : PodString(str.c_str(), str.length()){ }

 bool IsValid() const noexcept{
        return ( m_value != Invalid()  );
  }

 operator  bool()  const noexcept {
        return   IsValid();
 }

 operator  value_type()  const noexcept {
        return  m_value;
 }
 operator  const char* () const noexcept {
  return reinterpret_cast<const char *>  (&m_value) ;
 }
 operator std::string () const noexcept {
       return reinterpret_cast<const char *>  (&m_value) ;
 }
private:
  static constexpr value_type  NullValue()  { return {} ;  }
  static constexpr value_type  Invalid() { return NullValue(); }
private:
  value_type  m_value{ };

};
}//PodStringImpl
using String_7 = PodStringImpl::PodString<7> ;
using String_15 = PodStringImpl::PodString<15> ;
using String_23 = PodStringImpl::PodString<23> ;
using String_31 = PodStringImpl::PodString<31> ;

}
#endif  /* CPP_TOOLS_POD_STRING_H */

/*
using namespace cpp_tools;
int main()
{

    String_7 value24 ("12345678");
  std::string v24 = value24;
  bool bv24 = value24;

 String_7 value1 ("SPY", 3);
 std::string v11 = value1;
  const char* v12 = value1;
  std::uint64_t v13 = value1;

  String_7 value2 ("SPY");
  String_7 value3 (std::string("SPY"));

  bool b1 = value1.IsValid(); bool b2 = value2.IsValid(); bool b3 = value3.IsValid();
  bool isArv1 = value1.IsArithmetic();

  String_15 value8("1"); String_15 value9("3");
  bool isArv8 = value8.IsArithmetic();
  String_15::value_type  im1 = value8;
  String_15::value_type im2 = value9;
  bool con12 = (im1 > im2);
  auto im3 = im1 + im2;

  String_23 value23("123456789123456789");
 std::string v23 = value23;
 bool b23 = value23;
bool isArv7 = String_7::IsArithmetic();
bool isArv15 = String_15::IsArithmetic();
bool isArv23 = String_23::IsArithmetic();


 return 0;
}
*/
```