

Integer Types In C and C++

By Jack Klein

Contents

Introduction

- The C and C++ Boolean Types
- Three Types of Char
- The Short Int Types
- The Int Types
- The Long Int Types
- The Long Long Int Types (C Only)
- Common Problems With Integer Types
- The Usual Integer Conversions *Coming Soon*
- Values In `<limits.h>`
- Two C Programs To Display Integer Type Information
- Two C++ Programs To Display Integer Type Information

Introduction

You would think that the basic integer types provided by the C and C++ languages wouldn't cause an much confusion as they do. Almost every day there are posts in the C and C++ newsgroups which show that many newcomers do not understand them. Some experienced programmers who are only familiar with one platform do not understand them either.

The most common source of confusion are the sizes of the integer types, and the range of values which they can hold. That is because the languages leave many features of the integer types *implementation-defined*, meaning that it is up to the particular compiler to determine their exact specifications. C and C++ do set minimum requirements for each of the integer types, but the compiler is free to exceed these limits.

Each compiler is required to document its implementation. This information should be available in the printed manuals, online help, or man pages which come with the compiler.

In addition, there is a required standard header named `<limits.h>` (`<climits>` in newer C++ compilers) that provides information about the integer types that can be used in your programs at run time. A compiler is not required to provide a header like `<limits.h>` as a readable text file, but I do not know of any compilers which do not.

There are programs on this page to display the information that this file contains.

The C and C++ Boolean Types

Both C and C++ now include a built-in *Boolean* type. For those unfamiliar with term, a Boolean type is a data type that can only hold two values, either **0** and **1** or **true** and **false**. The two languages implement their Boolean type a little differently, so here is a discussion of the C Boolean type, and here is one for the C++ Boolean type.

Integer Types In C and C++

By Jack Klein

The C Boolean Type

The 1999 update to the C language standard, commonly called **C99**, adds a Boolean type to C.

Three Types Of Char

C and C++ define three types of char:

- signed char
- unsigned char
- "plain" char (default char without *signed* or *unsigned*)

These three types are collectively known as *character types* and they have some special properties that no other data types have.

- Each of the character types occupies exactly one byte. That is, `sizeof(char) == 1`, by definition in C and C++, and always will.
- The header `<limits.h>` contains a macro, **CHAR_BIT**, that expands to an integer constant specifying the number of bits in the character types.
- There is a third type in addition to *signed* and *unsigned*. It is up to the compiler to decide whether "plain" char or default char is signed or unsigned.

These properties give rise to some common questions:

Q: *If `sizeof(char) == 1`, doesn't that mean that **CHAR_BIT** is always 8?*

A: The value of **CHAR_BIT** is required to be *at least* 8. Almost all modern computers today use 8 bit bytes (technically called *octets*, but there are still some in production and use with other sizes, such as 9 bits. Also some processors (especially **D**igital **S**ignal **P**rocessors) cannot efficiently access memory in smaller pieces than the processor's word size. There is at least one DSP I have worked with where **CHAR_BIT** is 32. The char types, short, int and long are all 32 bits.

The one byte which is `sizeof(char)` meets the C and C++ definition of a byte, which is the amount of memory required to store a character. It does not necessarily correspond to a machine byte.

Q: *Why is it up to the compiler to decide whether the default char type is signed or unsigned?*

A: This is just the way that the C language evolved. By the time a standard for the C language was being developed there were compilers that implemented ordinary char as signed, and others as unsigned. Some programs for these compilers depended on that choice. If the standard required one choice or the other it would break a lot of existing, working code.

On all compilers, "plain" char must have exactly the same representation and interpretation as either signed or unsigned char.

Many compilers offer a command line or IDE option to set default char to either signed or unsigned for compatibility with code developed on either type of system.

Q: *What other differences are there between the three types of char?*

Integer Types In C and C++ By Jack Klein

A: All three types of char are different types. A pointer to one type of char cannot be assigned to a pointer to another char type without a cast. This is also true when passing a pointer as an argument to a function.

Q: *What other special properties do the char types have?*

A: Every bit in an object of unsigned character types contributes to its value. There are no unused or padding bits, and every possible combinations of bits represents a valid value for an unsigned char. There is no other data type in C or C++ that guarantees this to be true.

Also every object which program can legally access can be accessed as an array of unsigned chars. There are no illegal or trap representations for an unsigned char.

Q: *What range of values can the character types hold?*

A: Signed char can hold all values in the range of **SCHAR_MIN** to **SCHAR_MAX**, defined in <limits.h>. **SCHAR_MIN** must be -127 or less (more negative), and **SCHAR_MAX** must be 127 or greater. Note that many compilers for processors which use a 2's complement representation support **SCHAR_MIN** of -128, but this is not required by the standards.

Unsigned char can hold values between 0 and **UCHAR_MAX** inclusive, which is required to be at least 255. If **CHAR_BIT** is greater than 8, **UCHAR_MAX** is required to be $2^{\text{CHAR_BIT}} - 1$. So an implementation which uses 9 bit chars can fit values from 0 to 511 in an unsigned char.

Finally default or "plain" char can hold all the values between **CHAR_MIN** and **CHAR_MAX** inclusive. If char is signed by default **CHAR_MIN** is equal to **SCHAR_MIN** and **CHAR_MAX** is equal to **SCHAR_MAX**. Otherwise the default char type is unsigned and **CHAR_MIN** is 0 and **CHAR_MAX** is equal to **UCHAR_MAX**.

The Short Int Types

There are two types of short int, signed and unsigned. If neither is specified the short is signed. The "int" in the declaration is optional. All 6 of the following declarations are correct:

short x;	x is a signed short int
short int x;	x is a signed short int
signed short x;	x is a signed short int
signed short int x;	x is a signed short int
unsigned short x;	x is an unsigned short int
unsigned short int x;	x is an unsigned short int

The range of the short int types:

- A signed short can hold all the values between **SHRT_MIN** and **SHRT_MAX** inclusive. **SHRT_MIN** is required to be -32767 or less, **SHRT_MAX** must be at least 32767. Again, many 2's complement implementations will define **SHRT_MIN** to be -32768 but this is not required.

Integer Types In C and C++

By Jack Klein

- An unsigned short can hold all the values between 0 and **USHRT_MAX** inclusive. **USHRT_MAX** must be at least 65535.
- The short types must contain **at least** 16 bits to hold the required range of values.

On many (but not all) C and C++ implementations, a short is smaller than an int. Programs which need to have very large arrays of integer values in memory, or store very large numbers of integers in files might save memory or storage space by using short in place of int if both of the conditions below are met:

1. Short is truly smaller than int on the implementation.
2. All of the required values can fit into a short.

NOTE: On some processor architectures code to manipulate shorts can be larger and slower than corresponding code which deals with ints. This is particularly true on the Intel x86 processors executing 32 bit code, as in programs for Windows (NT/95/98), Linux, and other UNIX derivatives. Every instruction which references a short in such code is one byte larger and usually takes extra processor time to execute.

The Int Types

An int was originally intended to be the "natural" word size of the processor. Many modern processors can handle different word sizes with equal ease. It is int which causes the greatest confusion. Some people are certain that an int has 16 bits and sizeof(int) is 2. Others are equally sure that an int has 32 bits and sizeof(int) is 4.

Who is right? On any given compiler, one or the other could be right. On some compilers, both would be wrong. I know of one compiler for a 24 bit **DSP** where an int has 24 bits. The actual range of values which the signed and unsigned int types can hold are:

- A signed int can hold all the values between **INT_MIN** and **INT_MAX** inclusive. **INT_MIN** is required to be -32767 or less, **INT_MAX** must be at least 32767. Again, many 2's complement implementations will define **INT_MIN** to be -32768 but this is not required.
- An unsigned int can hold all the values between 0 and **UINT_MAX** inclusive. **UINT_MAX** must be at least 65535. The int types must contain **at least** 16 bits to hold the required range of values.

NOTE: *The required ranges for signed and unsigned int are identical to those for signed and unsigned short. On compilers for 8 and 16 bit processors (including Intel x86 processors executing in 16 bit mode, such as under MS-DOS), an int is usually 16 bits and has exactly the same representation as a short. On compilers for 32 bit and larger processors (including Intel x86 processors executing in 32 bit mode, such as Win32 or Linux) an int is usually 32 bits long and has exactly the same representation as a long.*

Integer Types In C and C++

By Jack Klein

The Long Int Types

There are two types of long int, signed and unsigned. If neither is specified the long is signed. The "int" in the declaration is optional. All 6 of the following declarations are correct:

long x;	x is a signed long int
long int x;	x is a signed long int
signed long x;	x is a signed long int
signed long int x;	x is a signed long int
unsigned long x;	x is an unsigned long int
unsigned long int x;	x is an unsigned long int

The range of the long int types:

- A signed long can hold all the values between **LONG_MIN** and **LONG_MAX** inclusive. **LONG_MIN** is required to be -2147483647 or less, **LONG_MAX** must be at least 2147483647. Again, many 2's complement implementations will define **LONG_MIN** to be -2147483648 but this is not required.
- An unsigned long can hold all the values between 0 and **ULONG_MAX** inclusive. **ULONG_MAX** must be at least 4294967295. The long types must contain **at least** 32 bits to hold the required range of values.

On many (but not all) C and C++ implementations, a long is larger than an int. Today's most popular desktop platforms, such as Windows and Linux, run primarily on 32 bit processors and most compilers for these platforms use a 32 bit int which has the same size and representation as a long.

The Long Long Int Types (C Only)

64 bit processors have been readily available in workstations for several years, and they will be coming to desktop computers soon. These processors can address enough memory to have arrays with more elements than a 32 bit long can specify. Inexpensive disk drives in use today can hold a file which contains more bytes than a 32 bit offset can reach. The requirement for an integer type required to hold more than 32 bits is obvious.

The 1999 update to the ANSI/ISO C language standard has added a new integer type to C, one that is required to be at contain at least 64 bits. Included in this update are the new variable types signed and unsigned **long long** . The selected name comes from gcc and several other compilers which already provide this type as an extension. On 32 bit Windows compilers from Microsoft, Borland (and maybe others) this same extension has the name `__int64`.

While the C++ standard is quite new and most likely will not change for several years, it is almost certain that C++ compilers will add support for these types as well. The C++ compilers which come with the implementations mentioned above do.

There are two types of long long int, signed and unsigned. If neither is specified the long long is signed. The "int" in the declaration is optional. All 6 of the following declarations are correct:

Integer Types In C and C++ By Jack Klein

long long x;	x is a signed long long int
long long int x;	x is a signed long long int
signed long long x;	x is a signed long long int
signed long long int x;	x is a signed long long int
unsigned long long x;	x is an unsigned long long int
unsigned long long int x;	x is an unsigned long long int

The range of the long long int types:

- A signed long long can hold all the values between **LLONG_MIN** and **LLONG_MAX** inclusive. **LLONG_MIN** is required to be -9223372036854775807 or less, **LLONG_MAX** must be at least 9223372036854775807. Again, many 2's complement implementations will define **LLONG_MIN** to be -9223372036854775808 but this is not required.
- An unsigned long long can hold all the values between 0 and **ULLONG_MAX** inclusive. **ULLONG_MAX** must be at least 18446744073709551615. The long types must contain **at least** 64 bits to hold the required range of values.

Common Problems With Integer Types

A very common question from newcomers to C and C++ programming is how to validate input. They use the standard library functions and objects like scanf() or cin which work well if the input is well-formed and within range, but fail if the input is invalid. This can be a real problem when the input is coming from a user at a keyboard and the user types something unexpected!

The pages listed here shows two methods for bullet proof integer input, one using the standard library strtol() function, the other using a state machine.

Values In <limits.h>

The standard header <limits.h> contains macros which expand to values which allow a program to determine information about the ranges of values each integer type can hold at run time. Each of these types (except for "plain" char, that is char without a signed or unsigned in front of it) must be able to contain a minimum range of values.

Type	<limits.h> Constant	Minimum Value
signed char	SCHAR_MIN	-127
	SCHAR_MAX	127
unsigned char	UCHAR_MAX	0
		255
"plain" char	CHAR_MIN CHAR_MAX	(note 1)
signed short	SHRT_MIN	-32767
	SHRT_MAX	32767
unsigned short	USHRT_MAX	0
		65535

Integer Types In C and C++ By Jack Klein

signed int	INT_MIN INT_MAX	-32767 32767
unsigned int	UINT_MAX	0 65535
signed long	LONG_MIN LONG_MAX	-2147483647 2147483647
unsigned long	ULONG_MAX	0 4294967295
signed long long	LLONG_MIN LLONG_MAX	-9223372036854775807 9223372036854775807
unsigned long long	ULLONG_MAX	0 18446744073709551615

Note 1: On implementations where default "plain" is signed, CHAR_MIN is equal to SCHAR_MIN and CHAR_MAX is equal to SCHAR_MAX. If "plain" char is unsigned, CHAR_MIN is 0 and CHAR_MAX is equal to UCHAR_MAX.

Two C Programs To Display Integer Type Information

The reason I have included two programs is that the C99 standard update is recent, and there are no fully conforming C99 compilers at the time of this writing. So there is one version for C99 implementations if you have one, and another for other compilers. If your compiler has a command line option or IDE checkbox to select whether the default char type is signed or unsigned, try compiling and running these programs both ways.

A Program To Display Integer Type Information With Any C Compiler

```
.
#include <stdio.h>
#include <limits.h>

volatile int char_min = CHAR_MIN;

int main(void)
{
    printf("\n\n\n\n\n    Character Types\n");
    printf("Number of bits in a character: %d\n",
        CHAR_BIT);
    printf("Size of character types is %d byte\n",
        (int)sizeof(char));
    printf("Signed char min: %d max: %d\n",
        SCHAR_MIN, SCHAR_MAX);
    printf("Unsigned char min: 0 max: %u\n",
        (unsigned int)UCHAR_MAX);

    printf("Default char is ");
```

Integer Types In C and C++ By Jack Klein

```
if (char_min < 0)
    printf("signed\n");
else if (char_min == 0)
    printf("unsigned\n");
else
    printf("non-standard\n");

printf("\n    Short Int Types\n");
printf("Size of short int types is %d bytes\n",
    (int)sizeof(short));
printf("Signed short min: %d max: %d\n",
    SHRT_MIN, SHRT_MAX);
printf("Unsigned short min: 0 max: %u\n",
    (unsigned int)USHRT_MAX);

printf("\n    Int Types\n");
printf("Size of int types is %d bytes\n",
    (int)sizeof(int));
printf("Signed int min: %d max: %d\n",
    INT_MIN, INT_MAX);
printf("Unsigned int min: 0 max: %u\n",
    (unsigned int)UINT_MAX);

printf("\n    Long Int Types\n");
printf("Size of long int types is %d bytes\n",
    (int)sizeof(long));
printf("Signed long min: %ld max: %ld\n",
    LONG_MIN, LONG_MAX);
printf("Unsigned long min: 0 max: %lu\n",
    ULONG_MAX);

return 0;
}
```

A Program To Display Integer Type Information With A C99 Conforming Compiler.

If your complains about *_Bool* or *long long* when compiling this program it does not support these C99 data types.

```
#include <stdio.h>
#include <limits.h>

volatile int char_min = CHAR_MIN;

int main(void)
{
    printf("Size of Boolean type is %d byte(s)\n\n",
```

Integer Types In C and C++ By Jack Klein

```
(int)sizeof(_Bool));

printf("Number of bits in a character: %d\n",
    CHAR_BIT);
printf("Size of character types is %d byte\n",
    (int)sizeof(char));
printf("Signed char min: %d max: %d\n",
    SCHAR_MIN, SCHAR_MAX);
printf("Unsigned char min: 0 max: %u\n",
    (unsigned int)UCHAR_MAX);

printf("Default char is ");
if (char_min < 0)
    printf("signed\n\n");
else if (char_min == 0)
    printf("unsigned\n\n");
else
    printf("non-standard\n\n");

printf("Size of short int types is %d bytes\n",
    (int)sizeof(short));
printf("Signed short min: %d max: %d\n",
    SHRT_MIN, SHRT_MAX);
printf("Unsigned short min: 0 max: %u\n",
    (unsigned int)USHRT_MAX);

printf("Size of int types is %d bytes\n",
    (int)sizeof(int));
printf("Signed int min: %d max: %d\n",
    INT_MIN, INT_MAX);
printf("Unsigned int min: 0 max: %u\n",
    (unsigned int)UINT_MAX);

printf("Size of long int types is %d bytes\n",
    (int)sizeof(long));
printf("Signed long min: %ld max: %ld\n",
    LONG_MIN, LONG_MAX);
printf("Unsigned long min: 0 max: %lu\n",
    ULONG_MAX);

printf("Size of long long types is %d bytes\n",
    (int)sizeof(long long));
printf("Signed long long min: %lld max: %lld\n",
    LLONG_MIN, LLONG_MAX);
printf("Unsigned long long min: 0 max: %llu\n",
    ULLONG_MAX);
```

Integer Types In C and C++ By Jack Klein

```
    return 0;  
}
```

Two C++ Program To Display Integer Type Information

I have also included two versions of the C++ program to display information about the integer types. The ANSI/ISO International Standard for C++ has been around just about a year longer than the C99 update to C, but there are still a lot of older, pre-standard C++ compilers in use. So there are Standard C++ and pre-Standard C++ versions of this program. If your compiler has a command line option or IDE checkbox to select whether the default char type is signed or unsigned, try compiling and running this program both ways.

A Program To Display Integer Type Information With Older C++ Compilers

```
.  
#include <iostream.h>  
#include <limits.h>  
  
volatile int char_min = CHAR_MIN;  
  
int main(void)  
{  
    cout << "\n\n\n\n\n    Character Types"  
    << endl;  
    cout << "Number of bits in a character: " <  
    < CHAR_BIT << endl;  
    cout << "Size of character types is " <<  
    sizeof(char) << " byte" << endl;  
    cout << "Signed char min: " << SCHAR_MIN  
    << " max: " << SCHAR_MAX << endl;  
    cout << "Unsigned char min: 0 max: " <<  
    UCHAR_MAX << endl;  
  
    cout << "Default char is ";  
    if (char_min < 0)  
        cout << "signed";  
    else if (char_min == 0)  
        cout << "unsigned";  
    else  
        cout << "non-standard";  
    cout << endl;  
  
    cout << "\n    Short Int Types" <<  
    endl;  
    cout << "Size of short int types is " <<  
    sizeof(short) << " bytes" << endl;  
    cout << "Signed short min: " << SHRT_MIN
```

Integer Types In C and C++ By Jack Klein

```
<< " max: " << SHRT_MAX << endl;
cout << "Unsigned short min: 0 max: " <<
USHRT_MAX << endl;

cout << "\n      Int Types" << endl;
cout << "Size of int types is " <<
sizeof(int) << " bytes" << endl;
cout << "Signed int min: " << INT_MIN
<< " max: " << INT_MAX << endl;
cout << "Unsigned int min: 0 max: " <<
UINT_MAX << endl;

cout << "\n      Long Int Types" <<
endl;
cout << "Size of long int types is " <<
sizeof(long) << " bytes" << endl;
cout << "Signed long min: " << LONG_MIN
<< " max: " << LONG_MAX << endl;
cout << "Unsigned long min: 0 max: " <<
ULONG_MAX << endl;

return 0;
}
```

A Program To Display Integer Type Information Standard C++ Compilers

```
.
#include <iostream>
#include <climits>

using std::cout;
using std::endl;

volatile int char_min = CHAR_MIN;

int main(void)
{
    cout << "Size of boolean type is "
        << sizeof(bool) << " byte(s)"
        << "\n\n";

    cout << "Number of bits in a character: "
        << CHAR_BIT << "\n";
    cout << "Size of character types is "
        << sizeof(char)
        << " byte" << "\n";
    cout << "Signed char min: "
```

Integer Types In C and C++ By Jack Klein

```
<< SCHAR_MIN << " max: "  
<<SCHAR_MAX << '\n';  
cout << "Unsigned char min: 0 max: "  
<< UCHAR_MAX << '\n';  
  
cout << "Default char is ";  
  
if (char_min < 0)  
    cout << "signed";  
else if (char_min == 0)  
    cout << "unsigned";  
else  
    cout << "non-standard";  
cout << "\n\n";  
  
cout << "Size of short int types is "  
<< sizeof(short) << " bytes"  
<< '\n';  
cout << "Signed short min: "  
<< SHRT_MIN << " max: "  
<< SHRT_MAX << '\n';  
cout << "Unsigned short min: 0 max: "  
<< USHRT_MAX << "\n\n";  
  
cout << "Size of int types is "  
<< sizeof(int) << " bytes"  
<< '\n';  
cout << "Signed int min: "  
<< INT_MIN << " max: "  
<< INT_MAX << '\n';  
cout << "Unsigned int min: 0 max: "  
<< UINT_MAX << "\n\n";  
  
cout << "Size of long int types is "  
<< sizeof(long) << " bytes"  
<< '\n';  
cout << "Signed long min: " <<  
LONG_MIN << " max: "  
<< LONG_MAX << '\n';  
cout << "Unsigned long min: 0 max: "  
<< ULONG_MAX << endl;  
  
return 0;  
}
```