C Datatypes

Patrício R. Domingues

Departamento de Eng Informática

ESTG/IPLeiria

DISCLAIMER: This chapter is an adaptation of ”C (Basic) Data Types” - www.tenouk.com, ©

C data types

- Categories of C data types

1. Primitive types (ANSI C (C89)/ISO C (C90))
   - `char`, `short`, `int`, `float` and `double`
2. Primitive types added in C99
   - `long long`
3. User-defined types
   - `struct`, `union`, `enum` and `typedef`
4. Derived types
   - pointer, array and function pointer

C basic data types (#1)

Type	Size in Bits	Comments	Other Names
Primitive Types in ANSI C (C89)/ISO C (C90)			
char	≥ 8	▪ **sizeof()** will give the size in units of **char**s. ▪ need not be 8-bit ▪ The number of bits is given by the CHAR_BIT macro in the limits.h header. ▪ Integer operations can be performed portably only for the range: 0 ~ 127 (2^8 / 2).	—
signed char	Same as **char** but guaranteed to be signed	▪ Can store integers in the range: -128 ~ 127 (2^8) portably.	—
unsigned char	Same as **char** but guaranteed to be unsigned.	▪ Can store integers in the range: 0 ~ 255 (2^8) portably.	—

C basic data types (#2)

short	≥ 16, ≥ size of **char**	▪ Can store integers in the range: -32768 ~ 32767 (2^{16} / 2) portably. ▪ Reduce memory usage ▪ The resulting executable may be larger and probably slower as compared to using **int**.	**short int**, **signed short**, **signed short int**
unsigned short	Same as **short** but unsigned	▪ Can store integers in the range: 0 ~ 65535 (2^{16}) portably. • Used to reduce memory usage • The resulting executable may be larger and probably slower as compared to using **int**.	**unsigned short int**
int	≥ 16, ≥ size of **short**	▪ Basic signed integer type. ▪ Represent a typical processor's data size which is word-size ▪ An integral data-type. ▪ Can store integers in the range: -32768 ~ 32767 (2^{16} / 2) portably.	**signed**, **signed int**
unsigned int	Same as **int** but unsigned.	▪ Can store integers in the range: 0 ~ 65535 (2^{16}) portably.	**unsigned**

C basic data types (#3)

long	≥ 32, ≥ size of **int**	▪ long signed integer type. ▪ Can store integers in the range: -2147483648 ~ 2147483647 (2^{32} / 2) portably.	**long int**, **signed long**, **signed long int**
unsigned long	Same as **long** but unsigned	▪ Can store integers in the range: 0 ~ 4294967295 (2^{32}) portably.	**unsigned long int**
float	≥ size of **char**	▪ Used to reduce memory usage when the values used do not vary widely. ▪ The format used is implementation defined and unnecessarily obeys the IEEE 754 single-precision format. ▪ **unsigned** cannot be specified.	—
double	≥ size of **float**	▪ Typical floating-point data type used by processor. ▪ The format used is implementation defined and unnecessarily obeys the IEEE 754 double-precision format. ▪ **unsigned** cannot be specified.	—
long double	≥ size of **double**	▪ **unsigned** cannot be specified.	—

C basic data types (#4)

Type	Size in Bits	Comments	Other Names
Primitive Types added to ISO C (C99)			
`long long`	≥ 64, ≥ size of `long`	▪ Can store integers in the range: -2^{63} ~ +2^{63}-1 ▪ Portable	**long long int, signed long long, signed long long int**
`unsigned long long`	Same as `long long`, but `unsigned`.	▪ Can store integers in the range: 0 ~ +2^{64}-1 ▪ Portable	**unsigned long long int**

C basic data types (#5)

intmax_t	Signed integer types capable of representing any value of any signed integer type.	▪ typedef represents the signed integer type with largest possible range	—
uintmax_t	Unsigned integer types capable of representing any value of any unsigned integer type	▪ typedef represents the unsigned integer type with largest possible range	—

C basic data types (#6)

- Actual **size of integer types** varies by implementation
  - Windows, Linux, BSD etc.

- The only guarantee is that `long long` is not smaller than `long`, which is not smaller than `int`, which is not smaller than `short`

```
long long >= long >= int >= short
```

- `int` should be the integer type that the target processor is most efficient working with. For example, all types can be 64-bit

Size and pointer difference types (#1)

- `size_t` and `ptrdiff_t` types to represent memory-related quantities
- `size_t`
  - An unsigned integer type is used to represent the sizes of objects
  - Used as the return type of the `sizeof()` operator
  - `size_t` is used to represent the maximum size of any object (including arrays) in the particular implementation
  - The maximum size of `size_t` is provided via `SIZE_MAX`, a macro constant which is defined in the stdint.h header file
  - It is guaranteed to be at least `65535`
  - Use **`%zu`** as formatter in printf:

    ```
    printf("sizeof(int)=%zu\n", sizeof(int));
    ```

Size and pointer difference types (#2)

- `size_t` and `ptrdiff_t` types to represent memory-related quantities

- `ptrdiff_t` is used to represent the difference between pointers
  - Is the signed integer type of the result of subtracting two pointers

- The type's size is chosen so that it could store the maximum size of a theoretically possible array of any type

- On a 32-bit system `ptrdiff_t` will take 32 bits

- On a 64-bit one - 64 bits and it is portable

Properties of basic types

- Actual properties of basic types
- Examples
  - Max./Min. size of an **`int`**: `INT_MAX, INT_MIN`
  - Max./Min size of a **`float`**: `FLT_MAX, FLT_MIN`
- Basic arithmetic types, is provided via macro constants in two header files
  - [limits.h]() header
    - macros for integer types
    - Actual values depend on the implementation: compiler, platform, OS, etc.
  - [float.h]() header
    - macros for floating-point types

Fixed-width integer types (#1)

- C99/C11 standards include definitions of several integer types to enhance the portability of programs' portability

- Existing basic integer types are inadequate
  - their actual sizes are implementation defined and may vary across different systems
  - Not portable!

- All new types are defined in
  - inttypes.h
  - stdint.h

- Categories of the fixed-width types >>

Fixed-width integer types (#2)

- **Exact width** integer types
  - guaranteed to have the same number **N** of bits across all implementations. Included only if it is available in the implementation
  - `intN_t  (e.g., int8_t, int16_t, …)`
- **Least width** integer types
  - guaranteed to be the smallest type available in the implementation, that has at least specified number **N** of bits. Guaranteed to be specified for at least N=8, 16, 32, 64
  - `int_leastN_t  (e.g., int_least8_t, int_least16_t, …)`
- **Fastest** integer types
  - guaranteed to be the fastest integer type available in the implementation, that has at least specified number **N** of bits. Guaranteed to be specified for at least N=8, 16, 32, 64
  - `int_fastN_t  (e.g., int_fast8_t, int_fast16_t, …)`
- **Pointer** integer types
  - guaranteed to be able to hold a pointer
  - `intptr_t and uintptr_t`
- **Maximum width** integer types
  - guaranteed to be the largest integer type in the implementation.
  - `intmax_t and uintmax_t`

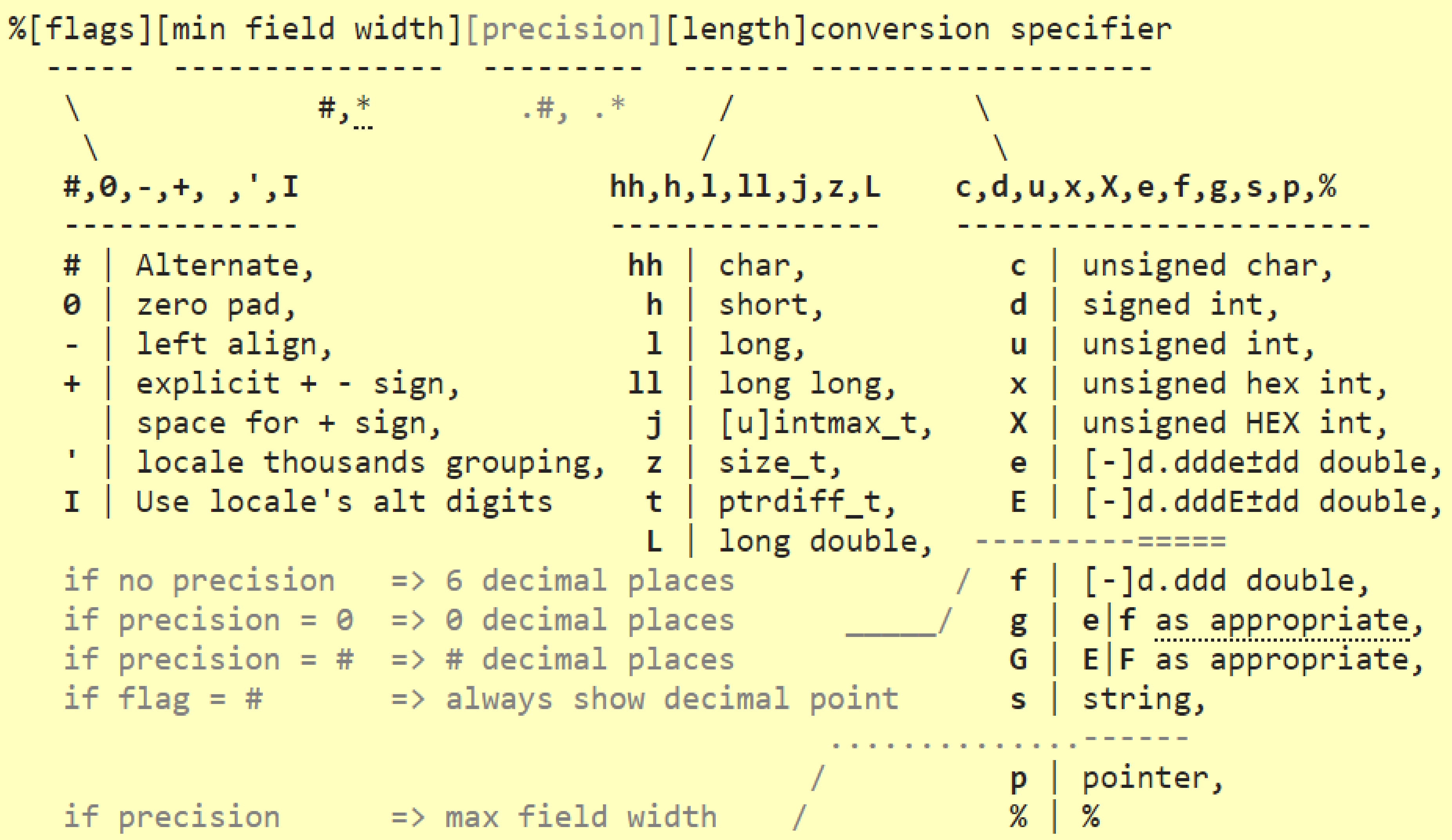
Fixed-width integer types (#3)

- Summary of the types and the interface to acquire the implementation details (**N** refers to the number of bits)

Type category	Signed types			Unsigned types		
	Type	**Min value**	**Max value**	**Type**	**Min value**	**Max value**
Exact width	`int`**N**`_t`	`INT`**N**`_MIN`	`INT`**N**`_MAX`	`uint`**N**`_t`	0	`UINT`**N**`_MAX`
Least width	`int_least`**N**`_t`	`INT_LEAST`**N**`_MIN`	`INT_LEAST`**N**`_MAX`	`uint_least`**N**`_t`	0	`UINT_LEAST`**N**`_MAX`
Fastest	`int_fast`**N**`_t`	`INT_FAST`**N**`_MIN`	`INT_FAST`**N**`_MAX`	`uint_fast`**N**`_t`	0	`UINT_FAST`**N**`_MAX`
Pointer	`intptr_t`	`INTPTR_MIN`	`INTPTR_MAX`	`uintptr_t`	0	`UINTPTR_MAX`
Maximum width	`intmax_t`	`INTMAX_MIN`	`INTMAX_MAX`	`uintmax_t`	0	`UINTMAX_MAX`

printf format specifiers (1)

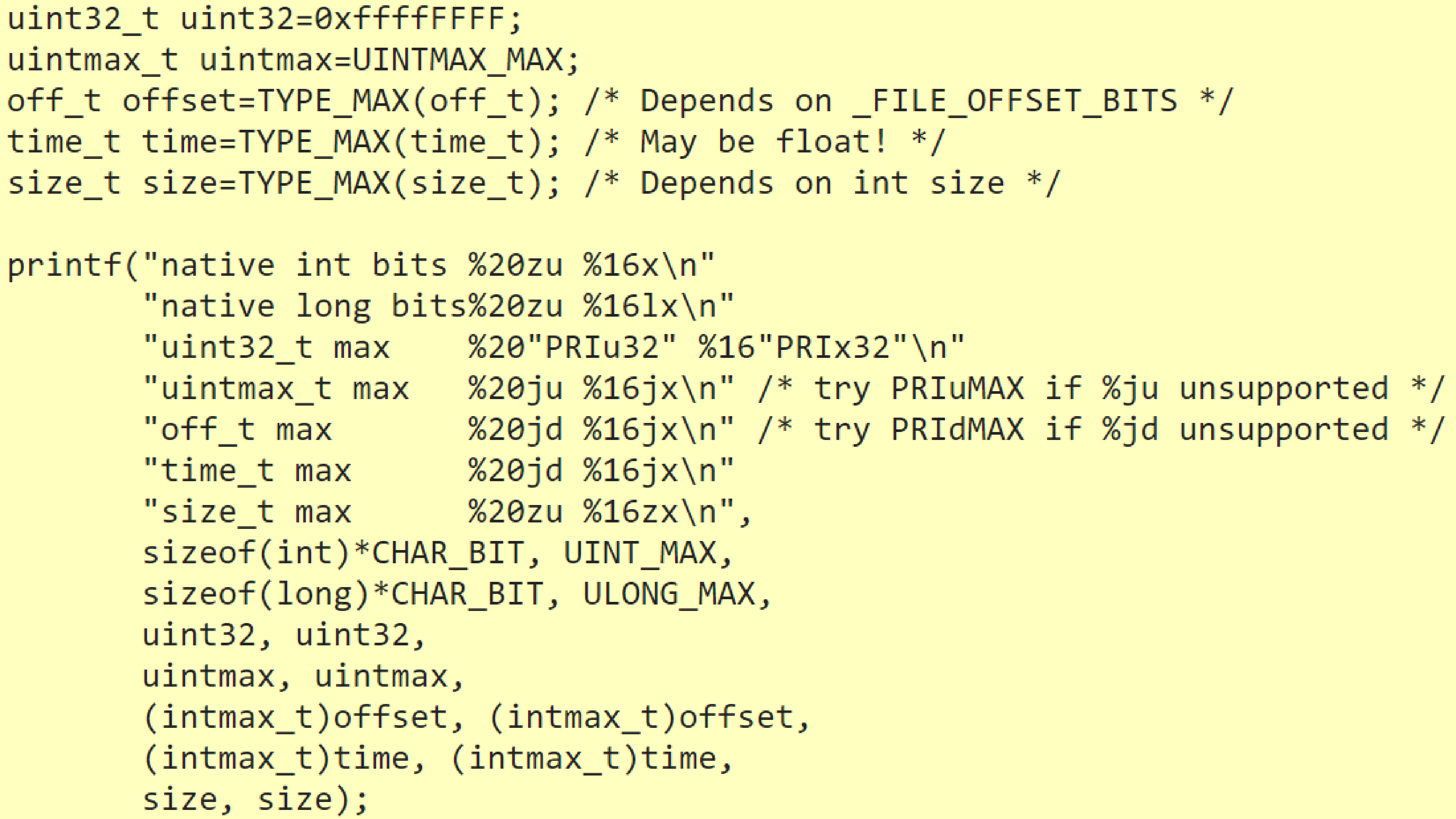
- Source: http://www.pixelbeat.org/programming/gcc/format_specs.html



```
%[flags][min field width][precision][length]conversion specifier
  -----  ---------------- ---------  ------ -------------------
   \             #,*        .#, .*     /             \
    \                                 /               \
   #,0,-,+, ,',I                 hh,h,l,ll,j,z,L    c,d,u,x,X,e,f,g,s,p,%
   -------------                 ---------------    -----------------------
   # | Alternate,                 hh | char,           c | unsigned char,
   0 | zero pad,                   h | short,          d | signed int,
   - | left align,                 l | long,           u | unsigned int,
   + | explicit + - sign,         ll | long long,      x | unsigned hex int,
     | space for + sign,           j | [u]intmax_t,    X | unsigned HEX int,
   ' | locale thousands grouping,  z | size_t,         e | [-]d.ddde±dd double,
   I | Use locale's alt digits     t | ptrdiff_t,      E | [-]d.dddE±dd double,
                                   L | long double,  --------=====
   if no precision   => 6 decimal places            /  f | [-]d.ddd double,
   if precision = 0  => 0 decimal places      _____/   g | e|f as appropriate,
   if precision = #  => # decimal places               G | E|F as appropriate,
   if flag = #       => always show decimal point      s | string,
                                          ..............------
                                         /             p | pointer,
   if precision      => max field width /              % | %
```

printf format specifiers (2)

- Source: http://www.pixelbeat.org/programming/gcc/int_types/



```
uint32_t uint32=0xffffFFFF;
uintmax_t uintmax=UINTMAX_MAX;
off_t offset=TYPE_MAX(off_t); /* Depends on _FILE_OFFSET_BITS */
time_t time=TYPE_MAX(time_t); /* May be float! */
size_t size=TYPE_MAX(size_t); /* Depends on int size */

printf("native int bits %20zu %16x\n"
       "native long bits%20zu %16lx\n"
       "uint32_t max    %20"PRIu32" %16"PRIx32"\n"
       "uintmax_t max   %20ju %16jx\n" /* try PRIuMAX if %ju unsupported */
       "off_t max       %20jd %16jx\n" /* try PRIdMAX if %jd unsupported */
       "time_t max      %20jd %16jx\n"
       "size_t max      %20zu %16zx\n",
       sizeof(int)*CHAR_BIT, UINT_MAX,
       sizeof(long)*CHAR_BIT, ULONG_MAX,
       uint32, uint32,
       uintmax, uintmax,
       (intmax_t)offset, (intmax_t)offset,
       (intmax_t)time, (intmax_t)time,
       size, size);
```

Macros for fprintf (#1)

- The **<inttypes.h>** defines macros for **`fprintf`** specifiers
- The fprintf() macros for signed integers are:

`PRId`**N** `PRIdLEAST`**N** `PRIdFAST`**N** `PRIdMAX PRIdPTR`

`PRIi`**N** `PRIiLEAST`**N** `PRIiFAST`**N** `PRIiMAX PRIiPTR`

Examples: `PRId8 (same as PRIi8); PRId64 (same as PRIi64)`

- The fprintf() macros for unsigned integers are:

`PRIo`**N** `PRIoLEAST`**N** `PRIoFAST`**N** `PRIoMAX PRIoPTR`

`PRIu`**N** `PRIuLEAST`**N** `PRIuFAST`**N** `PRIuMAX PRIuPTR`

`PRIx`**N** `PRIxLEAST`**N** `PRIxFAST`**N** `PRIxMAX PRIxPTR`

`PRIX`**N** `PRIXLEAST`**N** `PRIXFAST`**N** `PRIXMAX PRIXPTR`

Examples: `PRIo8 - octal for uint8_t; PRIX64 – hexadecimal for uint64_t`

- All macros are strings.
  - **Examples:** PRIX64 = `lX`; PRIi64=`li`; PRId64=`ld`

Example >>

Macros for fprintf (#2)

- Example
  - man inttypes.h

```
#include <stdio.h>
#include <inttypes.h>
int main(void){
   // This type always exists.
   uintmax_t i = UINTMAX_MAX;
   fprintf(stdout,"The largest integer value is %020"
               PRIxMAX "\n", i);
   return 0;
}
```

PRIxMAX is defined as "lx", x meaning ‘hexadecimal’

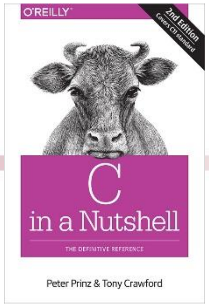
float, double and long double

- *floating point datatypes: size and limits*
  - `float`, `double` and `long double`
  - include <float.h>
    - float: FLT_MIN, FLT_MAX and FLT_DIG (precisão)
    - double: DBL_MIN, DBL_MAX and DBL_DIG
    - long double: LDBL_MIN, LDBL_MAX and LDBL_DIG

```
== FLOAT ==
sizeof(float): 4
FLT_MIN=0.000000,FLT_MAX=3.402823e+38,FLT_DIG=6
== DOUBLE ==
sizeof(double): 8
DBL_MIN=0.000000,DBL_MAX=1.797693e+308,DBL_DIG=15
== LONG DOUBLE ==
sizeof(long double): 16
LDBL_MIN=0.000000,LDBL_MAX=1.189731e+4932,LDBL_DIG=18
```

- C source: https://pastebin.com/LiRi9a9U

Example: precision



- Example (precision)

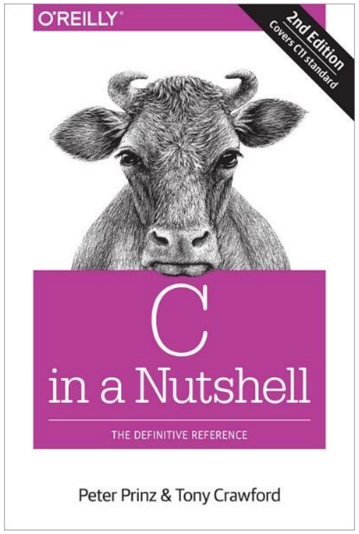
```
double d_var = 12345.6; // A variable of type double.
// Initializes the float variable
// with the value of d_var.
float f_var = (float)d_var;
printf("double: %18.10f\n", d_var);
printf("float: %18.10f\n", f_var);
printf("Rounding error is: %18.10f\n", d_var - f_var);
```

Output

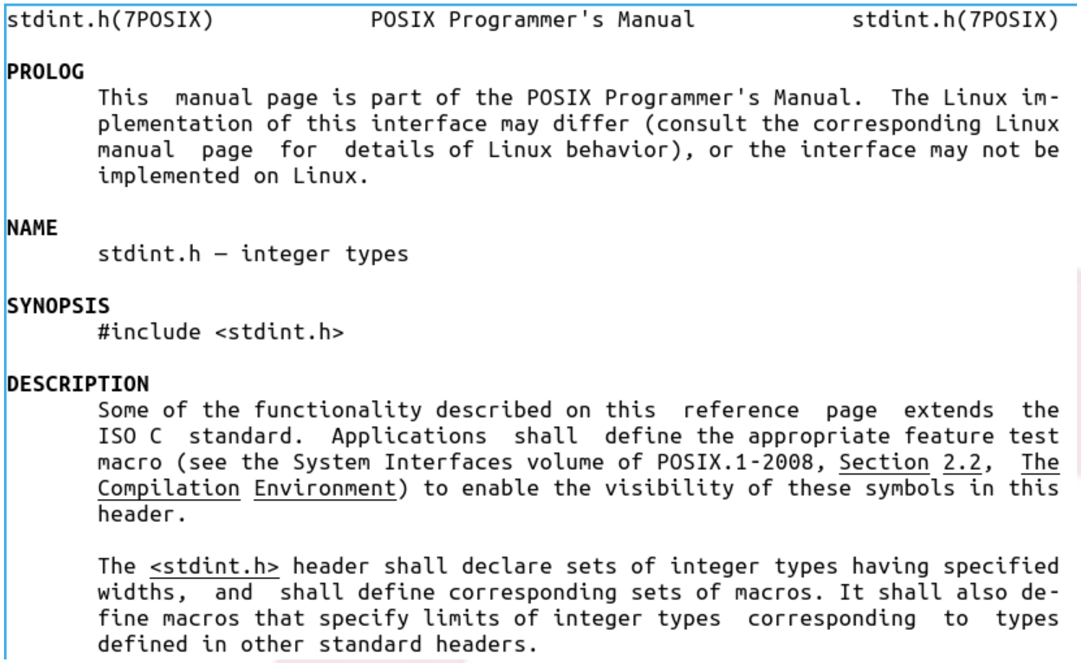
```
double: 12345.6000000000
float: 12345.5996093750
Rounding error is:  0.0003906250
```

References

- *"Types", Chapter 2*, C in a Nustshell, Peter Prinz, Tony Crawford, 2nd Edition, O'Reilly, 2015.



- www.tenouk.com
- man pages
  - man stdint.h
  - man inttypes.h



```
stdint.h(7POSIX)          POSIX Programmer's Manual          stdint.h(7POSIX)

PROLOG
       This  manual page is part of the POSIX Programmer's Manual.  The Linux im-
       plementation of this interface may differ (consult the corresponding Linux
       manual  page  for  details of Linux behavior), or the interface may not be
       implemented on Linux.

NAME
       stdint.h — integer types

SYNOPSIS
       #include <stdint.h>

DESCRIPTION
       Some of the functionality described on this  reference  page  extends  the
       ISO C  standard.  Applications  shall  define the appropriate feature test
       macro (see the System Interfaces volume of POSIX.1-2008, Section 2.2,  The
       Compilation Environment) to enable the visibility of these symbols in this
       header.

       The <stdint.h> header shall declare sets of integer types having specified
       widths,  and  shall define corresponding sets of macros. It shall also de-
       fine macros that specify limits of integer types  corresponding  to  types
       defined in other standard headers.
```