1. Definition:

Real types simulate real numbers. Real types are discrete whereas the set of real numbers is continuous. Real types are called floating-point numbers. The density of floating-point numbers is shown on a real number line in Figure 1.

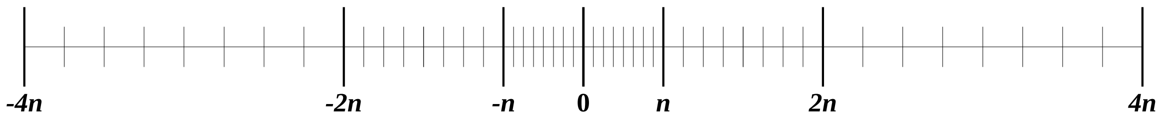

Figure 1. Density of floating-point numbers.

Sets: Each set is dependent on its representation.

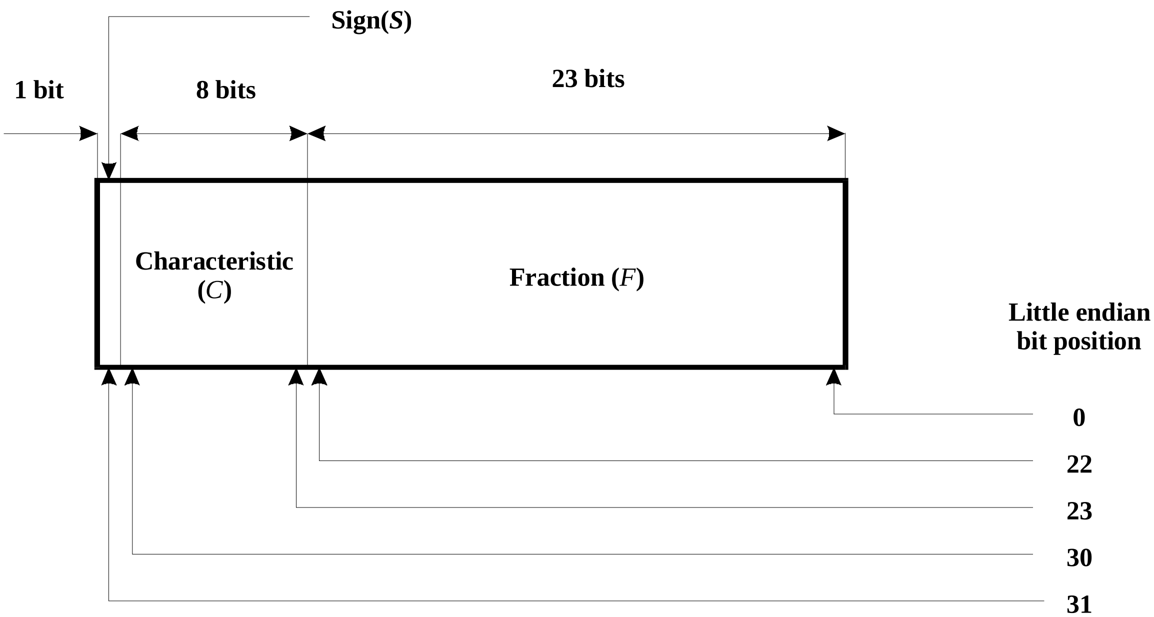

Figure 2. IEEE-754 single binary floating-point representation used to implement type **float**.

$$R = \{r \in R \mid -1^{s} \times 2^{c-b} \times 1.F\}, s \in \{0,1\}, c = 1 \le c < 254, b = 127, F = \sum_{k=1}^{23} f_k \times 2^{-k},\ f_k \in \{0,1\}$$

Figure 3. Set *R* contains the numbers that can be produced by the IEEE-754 single binary floating-point representation

$$R_f = \left\{ r \in R_f \mid s \times 2^{e} \times \sum_{k=0}^{24} f_k \times 2^{-k} \right\}, s \in \{-1,1\}, -126 \le e \le 127,\ f_k \in \{0,1\}$$

Figure 4. Set R_f

Set R_f is equivalent to set R. Set R_f is abstracted from set R.

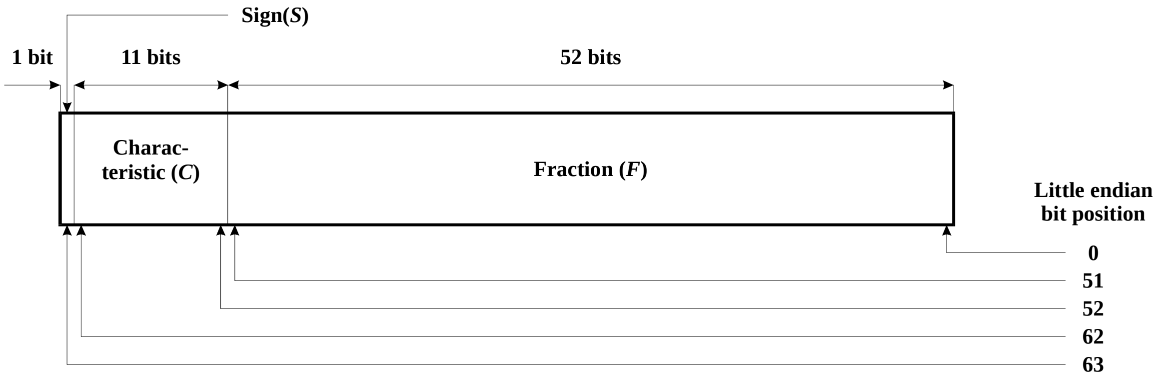

Figure 5. IEEE-754 double binary floating-point representation used to implement type **double**.

$$R = \left\{ r \in R \mid -1^s \times 2^{c-b} \times 1.F \right\}, s \in \{0,1\}, c = 1 \le c < 2047 b = 1023, F = \sum_{k=1}^{52} f_k \times 2^{-k}, \ f_k \in \{0,1\}$$

Figure 6. Set *R* contains the numbers that can be produced by the IEEE-754 single binary floating-point representation

$$R_d = \left\{ r \in R_f \mid s \times 2^e \times \sum_{k=0}^{53} f_k \times 2^{-k} \right\}, s \in \{-1,1\}, -1022 \le e \le 1023, \ f_k \in \{0,1\}$$

Figure 7. Set R_d

Set R_d is equivalent to set R. Set R_d is abstracted from set R.

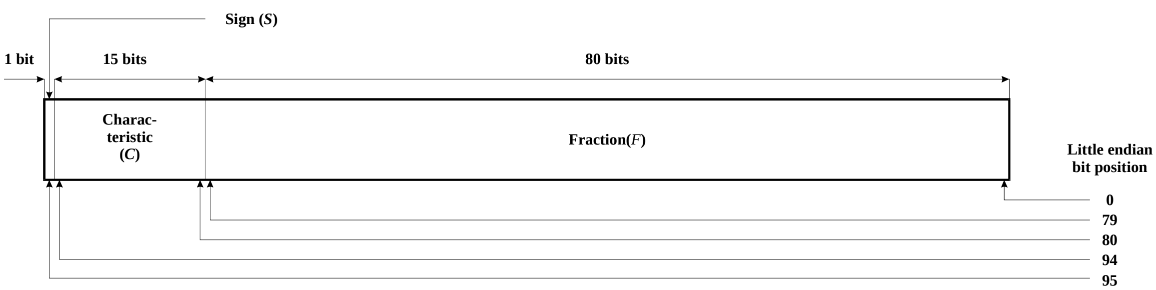

Figure 8. IEEE-754 double extended binary floating-point representation used to implement type **long double**.

$$R = \left\{ r \in R \mid -1^s \times 2^{c-b} \times 1.F \right\}, s \in \{0,1\}, c = 1 \le c < 32{,}767 b = 16{,}383, F = \sum_{k=1}^{80} f_k \times 2^{-k}, \ f_k \in \{0,1\}$$

Figure 9. Set *R* contains the numbers that can be produced by the IEEE-754 single binary floating-point representation

$$R_{lf} = \left\{ r \in R_f \mid s \times 2^e \times \sum_{k=0}^{53} f_k \times 2^{-k} \right\}, s \in \{-1,1\}, -1022 \le e \le 1023, \ f_k \in \{0,1\}$$

Figure 10. Set R_d

Set R_{lf} is equivalent to set R. Set R_{lf} is abstracted from set R.

2. Declarations:

declarations:
real-declaration-list **;**

real-declaration-list:
real-declaration
real-declaration-list **,** *real-declaration*

real-declaration:
real-declaration-specifier-sequence real-variable-name real-initialization$_{opt}$

real-declaration-specifier-sequence:
real-declaration-specifier
real-declaration-specifier-sequence real-declaration-specifier

real-declaration-specifier:
storage-class-specifier
real-type-specifier

storage-class-specifier:
auto
register
static
extern

real type-specifier:
float
double
long double

real-variable-name:
identifier

real-initialization:
= *assignment-expression*
(*assignment-expression* **)**

Examples:
float *f***;**
double *d***;**
long double *ld***;**

3. Constants:

Floating-point constants may be written with a decimal point, a signed exponent, or both. A floating-point constant is always interpreted to be in decimal radix. C++ allows a suffix letter (*floating-suffix*) to designate constants of types **float**, and **long double**. Without a suffix, the type of the constant is **double**.

floating-constant:
digit-sequence exponent floating-suffix$_{op}$*t*
dotted-digits exponent$_{opt}$ *floating-suffix*$_{op}$*t*

floating-suffix:

f | F | l | L

exponent:
E *sign-part*$_{opt}$ *digit-sequence*
e *sign-part*$_{opt}$ *digit-sequence*

sign-part:
+ | -

dotted-digits:
digit-sequence **.**
digit-sequence **.** *digit-sequence*
. *digit-sequence*

digit:
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Examples:

Constant	**Type**	**Description**
0.	**double**	0
3e1	**double**	30
3.14159	**double**	π
.0	**double**	0
1.0E-3	**double**	0.001
1e-3	**double**	0.001
1.0	**double**	1
.00034	**double**	3.4×10^{-4}
2e+9	**double**	2,000,000,000
1.0f	**float**	1
1.0e67L	**long double**	1×10^{67}
0E1L	**long double**	0×10^{1}

4. **Operations:** Operations on real types consist of the standard arithmetic operations of addition, subtraction, multiplication, and division. The **<***cmath***>** library also provides a rich set of useful operations primarily on real types.

4.1. Standard arithmetic operations.

Operation	**Operator**
Multiplication	*****
Division	**/**
Addition	**+**
Subtraction	**-**
Less than	**<**
Less than or equal	**<=**
Greater than	**>**
Greater than or equal	**>=**
Equality	**==**
Inequality	**!=**

Table 1. Real operations

4.2. **<***cmath***>** library**.** Selected functions from the **<***cmath***>** library.

Declaration	**Description**	**Example**
int *abs***(int** *x***);**	Function *abs***(***x***)** returns the absolute value of its integer argument *x*.	**int** *x***=-5;** *cout* **<<** *abs***(***x***);** Output 5
long *labs***(long int x);**	Function *labs***(***x***)** returns the absolute value of its integer argument *x*.	**long int** *x***=-5;** *cout* **<<** *labs***(***x***);** Output 5
double *ceil***(double** *x***);**	Function *ceil***(***x***)** returns the smallest floating-point number not less than *x* whose value is an exact mathematical integer.	**double** *x***=5.5;** *cout* **<<** *ceil***(***x***);** Output 6
double *floor***(double** *x***);**	Function *floor***(***x***)** returns the largest floating-point number not greater than *x* whose value is an exact mathematical integer.	**double** *x***=5.5;** *cout* **<<** *floor***(***x***);** Output 5
double *pow***(double** *b***,double** *e***);**	Function *pow***(***b,e***)** returns b^e	**double** *b***=2.0,***e***=5.0;** *cout* **<<** *pow***(***b***,***e***);** Output 32
double *sqrt***(double** *x***);**	Function *sqrt***(***x***)** returns $\sqrt{x}$	**double** *x***=81.0;** *cout* **<<** *sqrt***(x);** Output 9
int *srand***(unsigned** *seed***);**	Function *srand* may be used to initialize the pseudo-random number generator that is used to generate successive values for calls to *rand*.	Program **p07** in Figure 9 illustrates how samples from the uniform distribution can be generated. Functions *srand* and *rand* are employed to initialize and produce the uniform distribution.
int *rand***(void);**	Successive calls to function *rand* return integer values in the range 0 to the largest possible value of type **int** that are the results of a pseudo-random-number generator.	Program **p07** in Figure 9 illustrates how samples from the uniform distribution can be generated. Functions *srand* and *rand* are employed to initialize and produce the uniform distribution.

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <ctime>
using namespace std;
int main()
{     time_t t;
      srand((unsigned)time(&t));      //Seed rand using the time of day
      for (int a=0;a<10;a++) {
            if (a%5==0) cout << endl;
            //-----------------------------------------------------------------
            //Print random samples from the uniform distribution
            //-----------------------------------------------------------------
            cout << " " << fixed << setprecision(4) << (double)rand()/RAND_MAX;
      }
      cout << endl;
      return 0;
}
```

Figure 9. Program **p07** illustrates the use of functions *srand* and *rand*.

```
 0.1340 0.5934 0.0614 0.5062 0.5890
 0.2081 0.1618 0.8826 0.1784 0.6333
```

Figure 10. Program **p07** output[1]

[1] Program **p07** produces different output each time it is invoked because the pseudo-random-number generator seed is different. The pseudo-random-number generator seed is different because it is an unsigned integer representing the time of day.

5. Example programs.

5.1. Program **p08** prints the amount by which the dollar is devalued for inflation rates of 3%, 5%, 7%, and 9%. A ten-year period is printed.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{     double w3=1.0,w5=1.0,w7=1.0,w9=1.0;
      cout << endl;
      cout << "Year";
      cout << " " << setw(6) << "3%";
      cout << " " << setw(6) << "5%";
      cout << " " << setw(6) << "7%";
      cout << " " << setw(6) << "9%";
      for (int y=1;y<=10;y++) {
            cout << endl;
            cout << setw(4) << y;
            cout << " " << fixed << setprecision(4) << w3;
            cout << " " << fixed << setprecision(4) << w5;
            cout << " " << fixed << setprecision(4) << w7;
            cout << " " << fixed << setprecision(4) << w9;
            w3*=1.03; w5*=1.05; w7*=1.07; w9*=1.09;
      }
      cout << endl;
      return 0;
}
```

Figure 11. Program **p08**.

```
Year     3%     5%     7%     9%
   1 1.0000 1.0000 1.0000 1.0000
   2 1.0300 1.0500 1.0700 1.0900
   3 1.0609 1.1025 1.1449 1.1881
   4 1.0927 1.1576 1.2250 1.2950
   5 1.1255 1.2155 1.3108 1.4116
   6 1.1593 1.2763 1.4026 1.5386
   7 1.1941 1.3401 1.5007 1.6771
   8 1.2299 1.4071 1.6058 1.8280
   9 1.2668 1.4775 1.7182 1.9926
  10 1.3048 1.5513 1.8385 2.1719
```

Figure 12. Program **p08** output

5.2. Program **p09** computes the future value of a sequence of fixed deposit in an interest bearing account. The user is prompted for the monthly deposit, annual percentage on the account and the term.

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int main()
{     cout << "Enter the monthly deposit. ";
      double R;
      cin >> R;
      cout << "Enter the Annual Percentage Rate (APR) on the account. ";
      double APR;
      cin >> APR;
      double i=APR/1200;
      cout << "i=" << fixed << setprecision(6) << i;
      cout << endl;
      cout << "Enter the number of years in the term. ";
      double y;
      cin >> y;
      int n=(int)floor(y*12+0.5);
      cout << "n=" << n << endl;
      double S=R*(pow(1+i,n)-1)/i;
      cout      << "The balance on the account after " << y << " years will be "
                << "$" << fixed << setprecision(2) << S << ".";
      cout << endl;
      return 0;
}
```

Figure 13. Program **p09**.

```
Enter the monthly deposit. 100
Enter the Annual Percentage Rate (APR) on the account. 9
i=0.007500
Enter the number of years in the term. 20
n=240
The balance on the account after 20.000000 years will be $66788.69.
```

Figure 14. Program **p09** output.

References:

1. Horstman and Budd; *Big C++*; Section 2.1, 2.2, 2.3, 2.4
2. Stroustrup; The C++ Programming Language, 3rd Ed. Section 4.5

Exercises:

1. Horstman and Budd; *Big C++*; p 70, R2.1
2. Horstman and Budd; *Big C++*; p 70, R2.2
3. Horstman and Budd; *Big C++*; p 70, R2.3
4. Write a program that given an initial distance , s_0 , and initial velocity, v_0 , a rate of acceleration, a , and the amount of time a body was accelerated, t , will compute the distance from the origin.

1. Write a program that will find the roots of a second order polynomial. Horstman and Budd; *Big C++*; p 70, R2.1