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.](image)

Sets: Each set is dependent on its representation.

![Figure 2. IEEE-754 single binary floating-point representation used to implement type float.](image)

\[ R = \{ r \in R \mid -1^s \times 2^{e-1023} \times 1.F \}, s \in \{0,1\}, c = 1 \leq c < 254, b = 127, F = \sum_{k=1}^{23} f_k \times 2^{-k}, f_k \in \{0,1\} \]  

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

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

*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`.

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

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

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

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`.

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

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

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

Figure 10. Set \( R_f \)

Set \( R_f \) is equivalent to set \( R \). Set \( R_f \) is abstracted from set \( R \).
2. **Declarations:**

   - **real-declaration-list:**
     - real-declaration-list
     - real-declaration
     - real-initializationopt

   - **real-declaration:**
     - real-declaration-specifier-sequence real-variable-name real-initializationopt

   - **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-suffixopt
     - dotted-digits exponentopt floating-suffixopt

   - **floating-suffix:**
Examples:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.</td>
<td>double</td>
<td>0</td>
</tr>
<tr>
<td>3e1</td>
<td>double</td>
<td>30</td>
</tr>
<tr>
<td>3.14159</td>
<td>double</td>
<td>π</td>
</tr>
<tr>
<td>.0</td>
<td>double</td>
<td>0</td>
</tr>
<tr>
<td>1.0E-3</td>
<td>double</td>
<td>0.001</td>
</tr>
<tr>
<td>1e-3</td>
<td>double</td>
<td>0.001</td>
</tr>
<tr>
<td>1.0</td>
<td>double</td>
<td>1</td>
</tr>
<tr>
<td>.00034</td>
<td>double</td>
<td>3.4×10⁻⁴</td>
</tr>
<tr>
<td>2e+9</td>
<td>double</td>
<td>2,000,000,000</td>
</tr>
<tr>
<td>1.0f</td>
<td>float</td>
<td>1</td>
</tr>
<tr>
<td>1.0e67L</td>
<td>long double</td>
<td>1×10⁶⁷</td>
</tr>
<tr>
<td>0E1L</td>
<td>long double</td>
<td>0×10¹</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplication</td>
<td>*</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
</tr>
<tr>
<td>Addition</td>
<td>+</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
</tr>
<tr>
<td>Less than</td>
<td>&lt;</td>
</tr>
<tr>
<td>Less than or equal</td>
<td>&lt;=</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt;</td>
</tr>
<tr>
<td>Greater than or equal</td>
<td>&gt;=</td>
</tr>
<tr>
<td>Equality</td>
<td>==</td>
</tr>
<tr>
<td>Inequality</td>
<td>!=</td>
</tr>
</tbody>
</table>

Table 1. Real operations
4.2. `<cmath>` library. Selected functions from the `<cmath>` library.

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

---

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.

```cpp
#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.

### Output Table

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

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.

```cpp
#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