FUNDAMENTALS OF C++

THE C++
CHARACTER SET, IDENTIFIERS
KEYWORDS, CONSTANTS
DATA TYPES, VARIABLES

The C++ Character Set

Consists of letters, digits, special characters, white spaces.

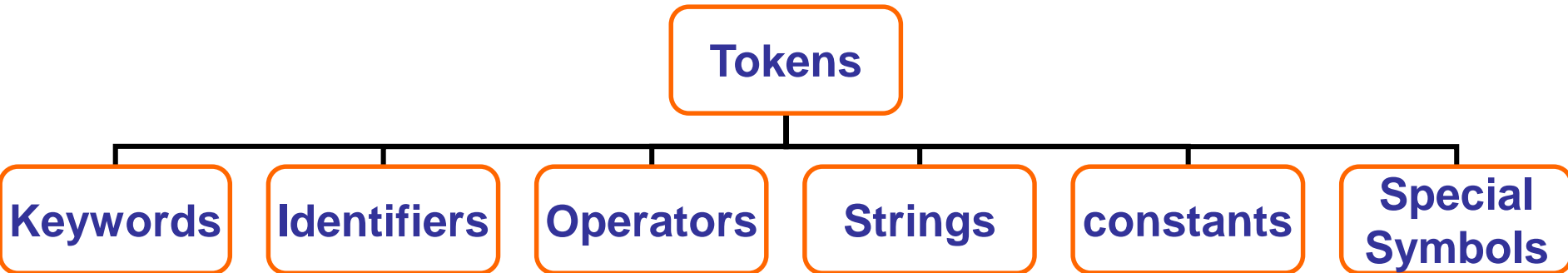**(i) Letters → 'a', 'b', 'c',………..z Or**

'A', 'B', 'C',……….Z

(ii) Digits → 0, 1, 2,…………….……9

(iii) Special characters → ;, ?, >, <, &,{, }, [,]……

(iv) White spaces → ex. New line (\n)

C++ Tokens

Tokens: Smallest individual units.

```
                          ┌──────────┐
                          │  Tokens  │
                          └────┬─────┘
   ┌──────────┬──────────┬─────┼──────────┬──────────┬──────────┐
┌────────┐ ┌──────────┐ ┌──────────┐ ┌─────────┐ ┌──────────┐ ┌──────────┐
│Keywords│ │Identifiers│ │Operators │ │ Strings │ │constants │ │ Special  │
└────────┘ └──────────┘ └──────────┘ └─────────┘ └──────────┘ │ Symbols  │
                                                               └──────────┘
```

Keywords ➔ words that are basically sequence of characters defined by a computer language that have one or more fixed meanings. They are also called as *reserve words*. Key words cannot be changed.

ex. int, float, do-while, if, else,…………..

Keywords

➢ Each keyword <u>has a predefined purpose</u> in the language.

➢ Do not use keywords as variable and constant names!!

➢ Some of the C/C++ keywords are

```
auto, bool, break, case, catch, class, char, const,
continue, do, default, delete, double, else,
extern, enum, false, float, for, friend, goto, if,
int, inline, long, namespace, new, operator,
private, protected, public, register, return,
short, static, struct, sizeof, switch, template,
this, throw, try, typedef, true, unsigned, virtual,
void, volatile, while …
```

C++ Tokens

Identifiers → **Example: user defined names like**

              int amount, float avg, …
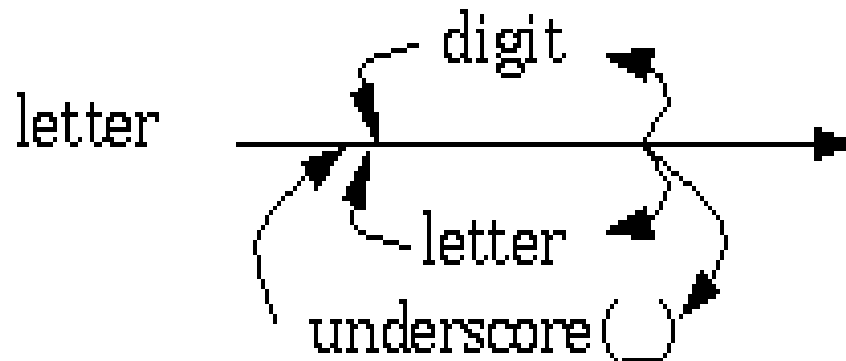
Operators → +, -, *, %, /, …
Strings → "Manipal"
Constants → -15, 10
Special Symbols → { } (, …

Identifiers

> An identifier is a <u>name for a variable, constant, function, etc.</u>

> It consists of a letter followed by any sequence of letters, digits, and underscores.

Identifiers

➢ An identifier is a <u>name for a variable, constant, function, etc.</u>

➢ A valid identifier is a sequence of one or more letters, digits or underscore character (_).

➢ Neither spaces nor punctuation marks or symbols can be part of an identifier

➢ Only letters, digits and underscore characters are valid

➢ variable identifiers always have to begin with a letter

➢ They can also begin with an underscore character (_), but this is usually reserved for compiler specific keywords or external identifiers.

Identifiers – rules

➢ They can not begin with a digit.

➢ The C/C++ is a <u>"case sensitive"</u> language.

➢ An identifier written in capital letters is not equivalent to another one with the same name but written in small letters.

➢ The "RESULT" variable is not the same as the "result" variable or the "Result" variable

➢ They cannot match any keyword of the C++ language or your compiler's specific ones since they could be confused with these.

Identifiers

➤ Examples of valid identifiers: `First_name`, `age`, `y2000`, `y2k`

➤ Examples of invalid identifiers: `2000y`

➤ Identifiers cannot have special characters in them. For example: `X=Y`, `J-20`, `~Ricky`,`*Michael` are invalid identifiers.

➤ Identifiers are case-sensitive.

For example: `Hello`, `hello`, `WHOAMI`, `WhoAmI`, `whoami` are unique identifiers.

Constants

✓ A value that IS NOT going to be changed during the execution of our program;

✓ Constant are specific values that are used in arithmetic expressions or assigned to variables, e.g. 2, 5, -10, 2.e+6, 3.14159, and so forth;

✓ Sometimes a constant represents truly a constant value in nature such as:

Pi 3.14159

speed of light 2.99792+E8 meters/sec

Constants
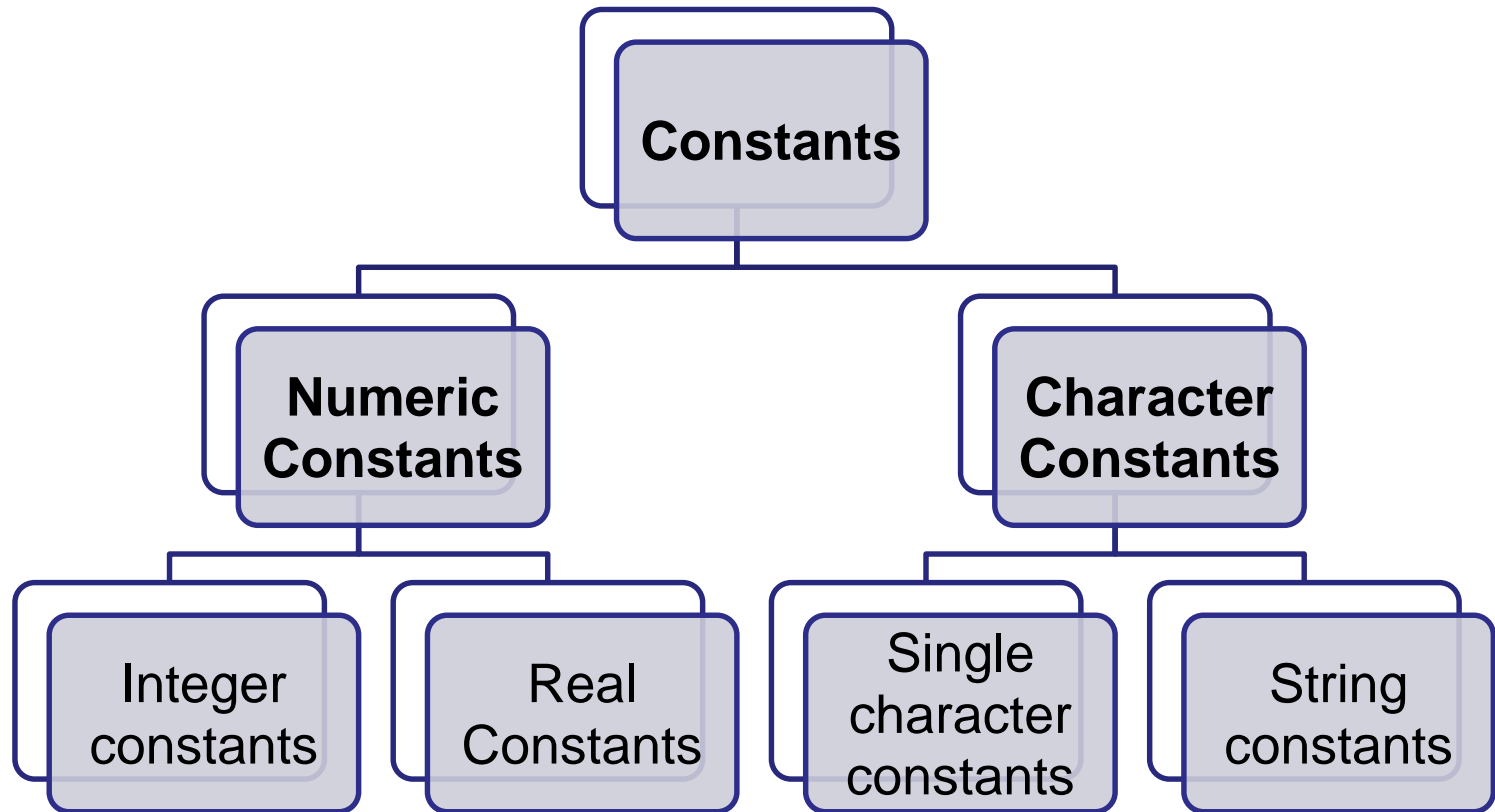
Numeric Constants

Integer Constants

Refers to a sequence of digits.

Decimal, Octal , Hexadecimal.

- ✓Decimal: set of digits 0 to 9, preceded by optional "–" or "+" sign

- ✓Octal: digits 0 to 7 with a leading "0"

- ✓Hexadecimal: digits 0 to 9, char A to F preceded by "0x"

E.g.: 143, -564, 0346, 0x34, 0x8AF

Floating Point Constant

Used to represent numbers with fractional part

E.g.; 213.45, .456,234.

Another form *mantissa e exponent*

   0.56e4, 3.12E4 , -4.6E-1

Character Constants

Single character

Single character with in a pair of single quote (' ') marks, having integer values known as ASCII values.

   E.g.: 'd', 't', '9'

String Constants

A sequence of characters enclosed in double quotes (" "). Characters may be letters, numbers, special characters and blank space.

E.g.: "hello", "2007", "T", "4+5"

Backslash character constants

Used in output functions

E.g.: '\n' new line, '\0' null char, '\a', '\b', '\t', '\"' …

Also known as escape characters.

Constant Qualifier

➢ Which helps us to associate an identifier with a constant value in your program;

➢ The advantage is that you can refer to the identifier any time you need to use the constant value instead of having to repeat writing the value;

➢ To declare a symbolic constant you must do it as follows:
   const *data type* *identifier* = *value*;

Constant Qualifier- example

Consider the Example below:

```
int main()
{
     float  area, perimeter;
     float  radius;
     const double Pi = 3.14159;
     area         = radius * radius * Pi;
     perimeter = 2 * Pi * radius;

     cout<<"Area equals "<<area;
     cout<<" Perimeter equals "<<perimeter;
     return 0;
}
```
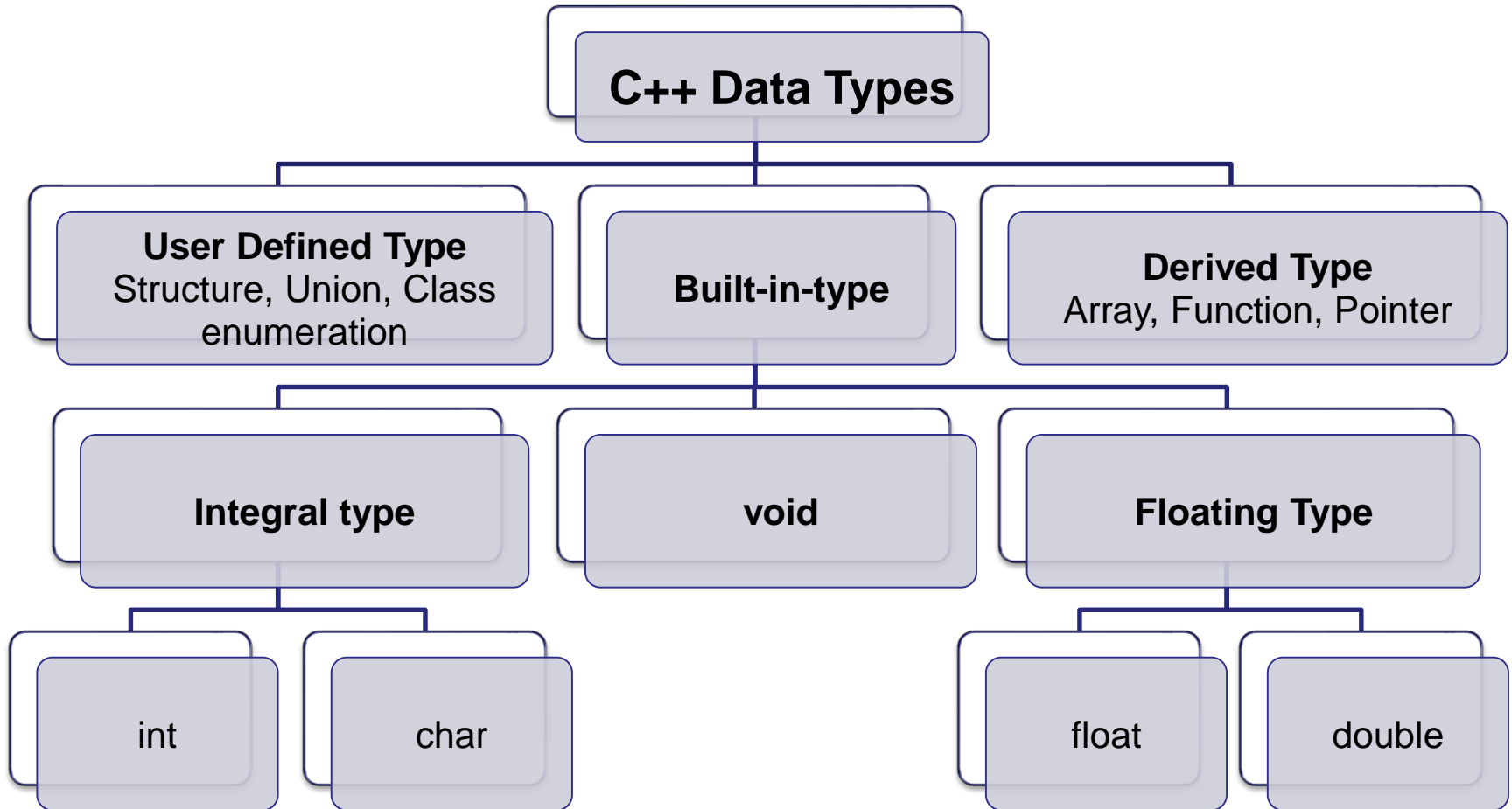
#define PI 3.14159

Variables

➢ A variable is a data name that may be used to store a data value.

➢ A variable may take different values at different times during execution.

➢ Variable name chosen by the programmer in a meaningful way.

➢ Each variable needs an identifier that distinguishes it from the others.

   (rules similar to identifier)

C++ data Types

```
                    C++ Data Types
                          |
        ┌─────────────────┼─────────────────┐
        |                 |                 |
  User Defined Type   Built-in-type    Derived Type
  Structure, Union, Class              Array, Function, Pointer
  enumeration
                          |
              ┌───────────┼───────────┐
              |           |           |
        Integral type    void    Floating Type
              |                       |
         ┌────┴────┐            ┌─────┴─────┐
        int      char        float      double
```

Data types

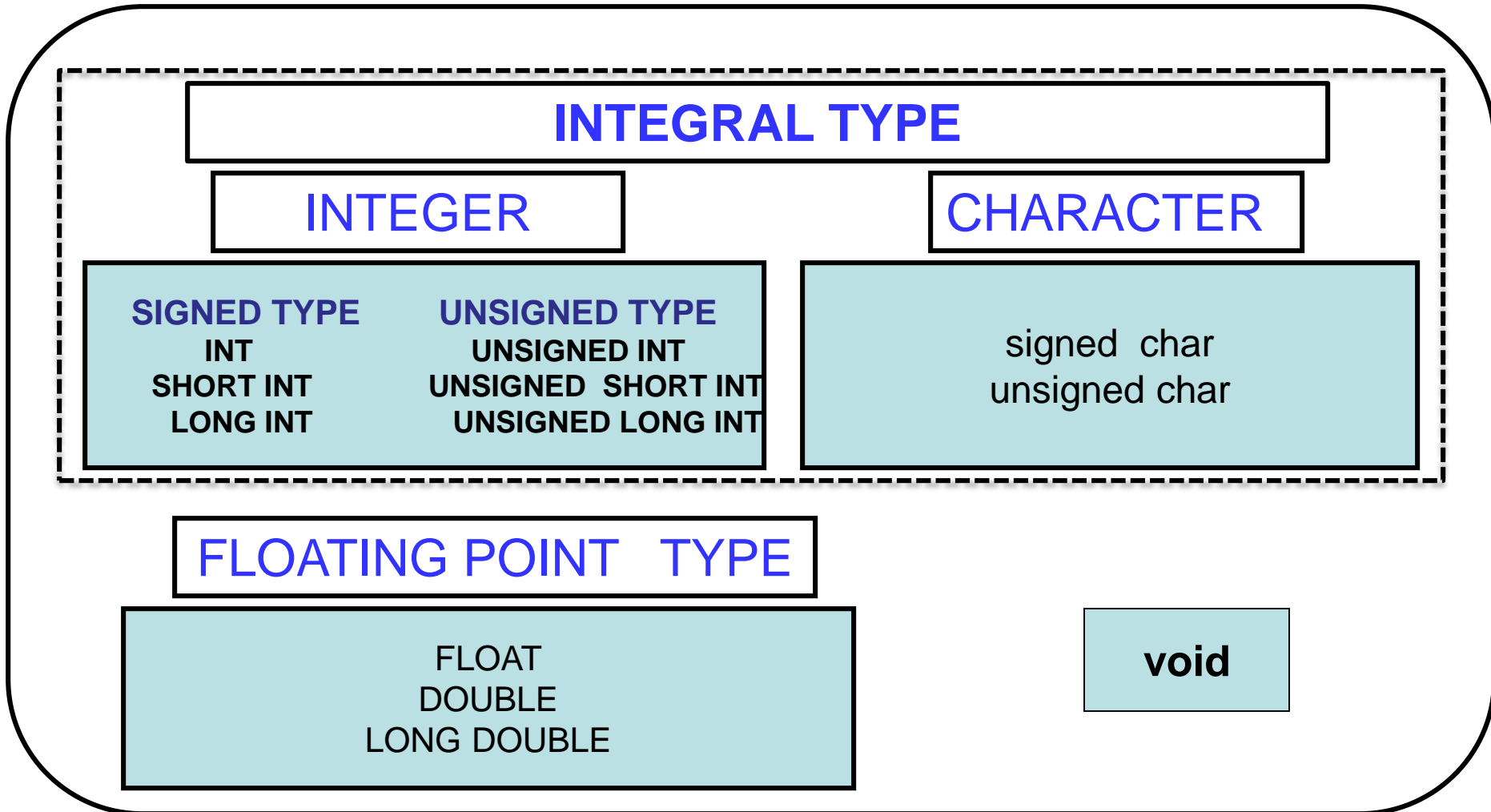**C++ Language is rich in its data types**

Mainly 4 classes of data types

  ➢ Primary (fundamental or Built-in type) data types,

  ➢ User defined Data types

  ➢ Derived Data types

  ➢ Empty data set.

The fundamental or Built-in data types fall into one of

 three categories

  ➢ Integer type

  ➢ Floating-point type

  ➢ Character type

Primary (built-in or Basic) Data types

INTEGRAL TYPE

INTEGER

CHARACTER

SIGNED TYPE
INT
SHORT INT
LONG INT

UNSIGNED TYPE
UNSIGNED INT
UNSIGNED SHORT INT
UNSIGNED LONG INT

signed char
unsigned char

FLOATING POINT TYPE

FLOAT
DOUBLE
LONG DOUBLE

void

Example

```cpp
int main()
{
    int a = 10, b = 20, i = 5, c;
    float f1, f2;

    c = a / b;
    f1 = a / b;
    f2 = i / 2;
    cout<<c<<"\n"<<f1<<"\n"<<f2;
    return 0;
}
```

OUTPUT

```
0
0
2
```

Example

```
 3  int main()
 4  {
 5      int a = 10, b = 20, i = 5;
 6      float f1, f2, f3;
 7
 8      f1 = a / (float)b;
 9      f2 = i / 2;
10      f3 = i / 2.0;
11      cout<<f1<<"\n"<<f2<<"\n"<<f3;
12      return 0;
13  }
```

OUTPUT

```
0.5
2
2.5
```

Integer Types

➢ The basic integer type is **int**
  ▪ The size of an **int** depends on the machine and the
    ✓ On PCs it is normally 16 or 32 bits
➢ Modifiers
  ▪ **short**: typically uses less bits
  ▪ **long**: typically uses more bits
  ▪ **unsigned**
  ▪ **signed**

| short int |
| int |
| long int |

➢ Different types allow programmers to use resources more efficiently.
➢ Standard arithmetic and relational operations are available for these types.

SIZE AND RANGE OF VALUES FOR A 16-BIT MACHINE (INTEGER TYPE)

	Type	Size (bits)	Range
short	short int or signed short int	8	-128 to 127
	unsigned int	8	0 to 255
integer	int or signed int	16	-32,768 to 32,767
	unsigned int	16	0 to 65,535
long	long int or signed long int	32	-2,147,483,648 to 2,147,483,647
	unsigned long int	32	0 to 4,294,967,295

```cpp
int main()
{
    cout<<"size of short int:"<< sizeof(short int)<<"\n";
    cout<<"size of char:"<<sizeof(char)<<"\n";
    cout<<"size of int:"<<sizeof(int)<<"\n";
    cout<<"size of float:"<<sizeof(float)<<"\n";
    cout<<"size of double:"<<sizeof(double)<<"\n";
    cout<<"size of long:"<<sizeof(long)<<"\n";
    cout<<"size of long double:"<<sizeof(long double)<<"\n";
    return 0;
}
```

OUTPUT

```
size of short int:2
size of char:1
size of int:4
size of float:4
size of double:8
size of long:4
size of long double:16
```

# SIZE AND RANGE OF VALUES FOR A 32/64-BIT MACHINE (INTEGER TYPE)

	Type	Size (bits)	Range
short	short int or signed short int		
	unsigned int		
integer	int or signed int		
	unsigned int		
long	long int or signed long int		
	unsigned long int		

Character Types

➢ Character type **`char`** is related to the integer types
➢ Modifiers unsigned and signed can be used
➢ char →1 byte(-128 to 127)
➢ signed char →1 byte(-128 to 127)
➢ unsigned char →1 byte(0 to 255)
➢ Characters are encoded using a scheme where an integer represents a particular character
➢ ASCII is the dominant encoding scheme
  (American Standard Code for Information Interchange)
  ▪ Examples
    ✓ `' '` encoded as 32   `'+'` encoded as 43
    ✓ **`'A'` encoded as 65**   **`'Z'` encoded as 90**
    ✓ **`'a'` encoded as 97**   **`'z'` encoded as 122**

Floating-Point Types

➢ Floating-point types represent real numbers

  ▪ Integer part

  ▪ Fractional part

➢ The number 108.1517 breaks down into the following parts

  ▪ 108 - integer part

  ▪ 1517 - fractional part

➢ C provides three floating-point types

  ▪ **float**

  ▪ **double**

  ▪ **long double**

float
double
long double
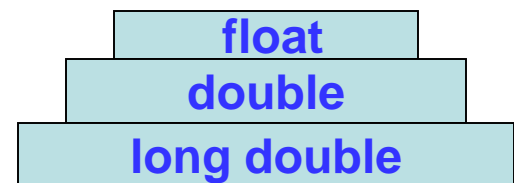
void

➢ 2 uses of void are

➢ To specify the return type of a function when it is not returning any value

➢ To indicate an empty argument list to a function

Boolean

- Logical or Boolean data- named after French Mathematician/philosopher George Boole

  – Consists of only two values: **true** and **false**

  – **Nonzero** number can be used to represent **true**

  – **Zero** is used to represent **false**

```cpp
int main()
{
    bool y = true;
    bool n = false;
    cout<<y<<"\n"<<n;
    return 0;
}
```

OUTPUT

```
1
0
```

Declaration of variables

➢ In order to use a variable in C++, we must first declare it.

➢ It does two things

   ✓Tells the compiler the variable name.

   ✓Specifies the type of data.

Primary Type declaration:

write the specifier of the desired data type (like int, char, float...) followed by a valid variable identifier.

i.e   *data-type V_1, V_2,…,V_n*

For example**:**

   int *a***;**

   float *mynumber, sum***;**

Declaration of variables

➢ The integer data types *short*, *long* and *int* can be either *signed* or *unsigned* depending on the range of numbers needed to be represented.

➢ Signed types can represent both *positive* and *negative* values, whereas unsigned types can only represent *positive values* (and zero).

Declaration of variables

➢ **Signed** and **unsigned** can be specified by using either the specifier *signed* or the specifier *unsigned* before the type name.

      For example:

           unsigned short int NumberOfSisters;

           signed int MyAccountBalance

➢ By default most compiler settings will assume the type to be signed(exception is char).

Declaration of variables

➢ Short and long can be used alone as type specifiers.

  ▪ The following two variable declarations are equivalent:

    ✓ short Year;

    ✓ short int Year;

➢ Signed and unsigned may also be used as standalone type specifiers.

  ▪ The following two declarations are equivalent:

    ✓ unsigned NextYear;

    ✓ unsigned int NextYear;

Declaration of variables

➢ Floating point:

     keywords →float, double, long double

➢ Character Type:

     keywords →char, unsigned char, signed char.

   Eg:

     double deviation;

     char p, q;

Assigning values to variables

➢ Values can be assigned using the assignment operator ' = '

  ✓ It is possible to assign at the time of declaration.

    data-type varaiable-name=constant

    Eg: **initial = 1;**

                area = 23.89;

                int final_value = 100, p = 20;

                char yes = 'm';

The process of giving initial values to variables➔ "*Initialization*".

  ✓ external & static variables initialized to zero by default.

const int *class_size* = 40;

const tells the compiler the value of variable *class_size* must not modified by the program. It can be used on the right hand side of any assignment statement.

Initialization of variables example

```cpp
int main ()
{   int a=5; // initial value = 5
    int b;
    b=2; // initial value = 2
    int result; // initial value undetermined
    a = a + 3;
    result = a - b;
    cout << result;
  return 0;
}
```