FUNDAMENTALS OF C++

Objective

- Become familiar with fundamental tokens and data types
- Write simple computer program in C++
- Use simple Output statements

Simple C++ program

```
// Display “This is my first C++ program”

// Single line comment
#include <iostream> // preprocessor directive
using namespace std;
int main( ) // Entry point for program execution
{
// block of statements: Begin
system (“clear” ); //Each Statement ends with ;
cout << “This is my first C++ program”;
return 0;
} // block of statements: End
```

Simple C++ Programs

```
// Multi lines comment
/*  Find the avg. of three marks and
display pass or fail */
#include <iostream>
using namespace std;
int main()
{
    cout << “Enter Roll Number and marks of three subjects”;
    int i_RollNo, i_marks1, i_marks2, i_marks3;
    float f_avg = 0;
    const float f_min = 35.0;
    cin >> i_marks1>> i_marks2>> i_marks3;
    f_avg = (i_marks1+i_marks2+i_marks3)/f_avg;
    if (f_avg < f_min)
        cout << RollNo <<“ fail”<< end;
    else
        cout << RollNo << “pass”<<end;
return 0;
}
```

Analogy between learning English Language and C++

- ✓ Steps in learning English Language:
  - Alphabets
  - Words
  - Sentence
  - Paragraphs

- ✓ Steps in learning C++
  - Alphabets, Digits, Special Symbols
  - Constants, Variables, Keywords
  - Instructions
  - Programs

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

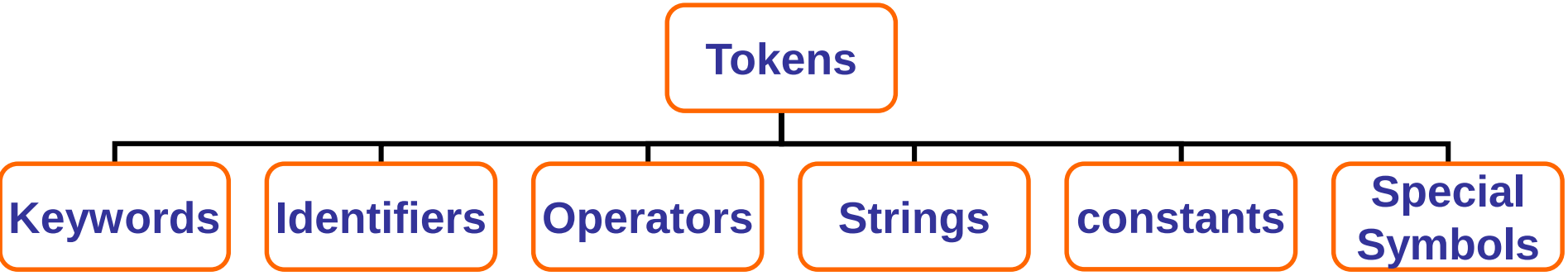
(ii) Digits → 0, 1, 2,……9

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

(iv) White spaces → ex. New line (\n)

C++ Tokens

Tokens: Smallest individual units.



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 has a predefined purpose 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”

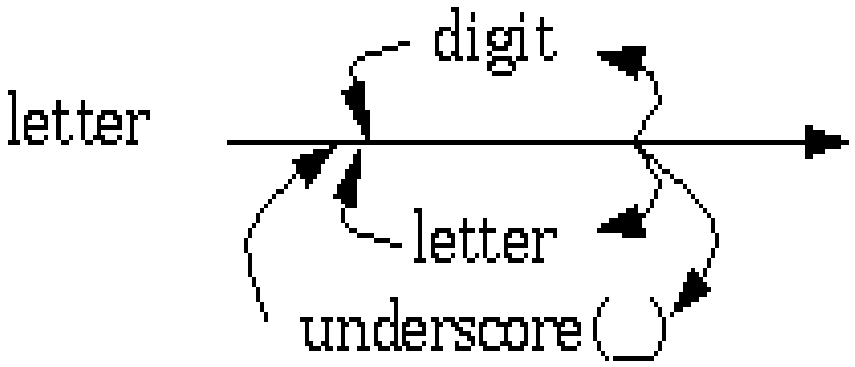
Constants → -15, 10

Special Symbols → { } (, …

Identifiers

- An identifier is a name for a variable, constant, function, etc.

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



Identifiers

- An identifier is a name for a variable, constant, function, etc.
- 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 "case sensitive" 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.

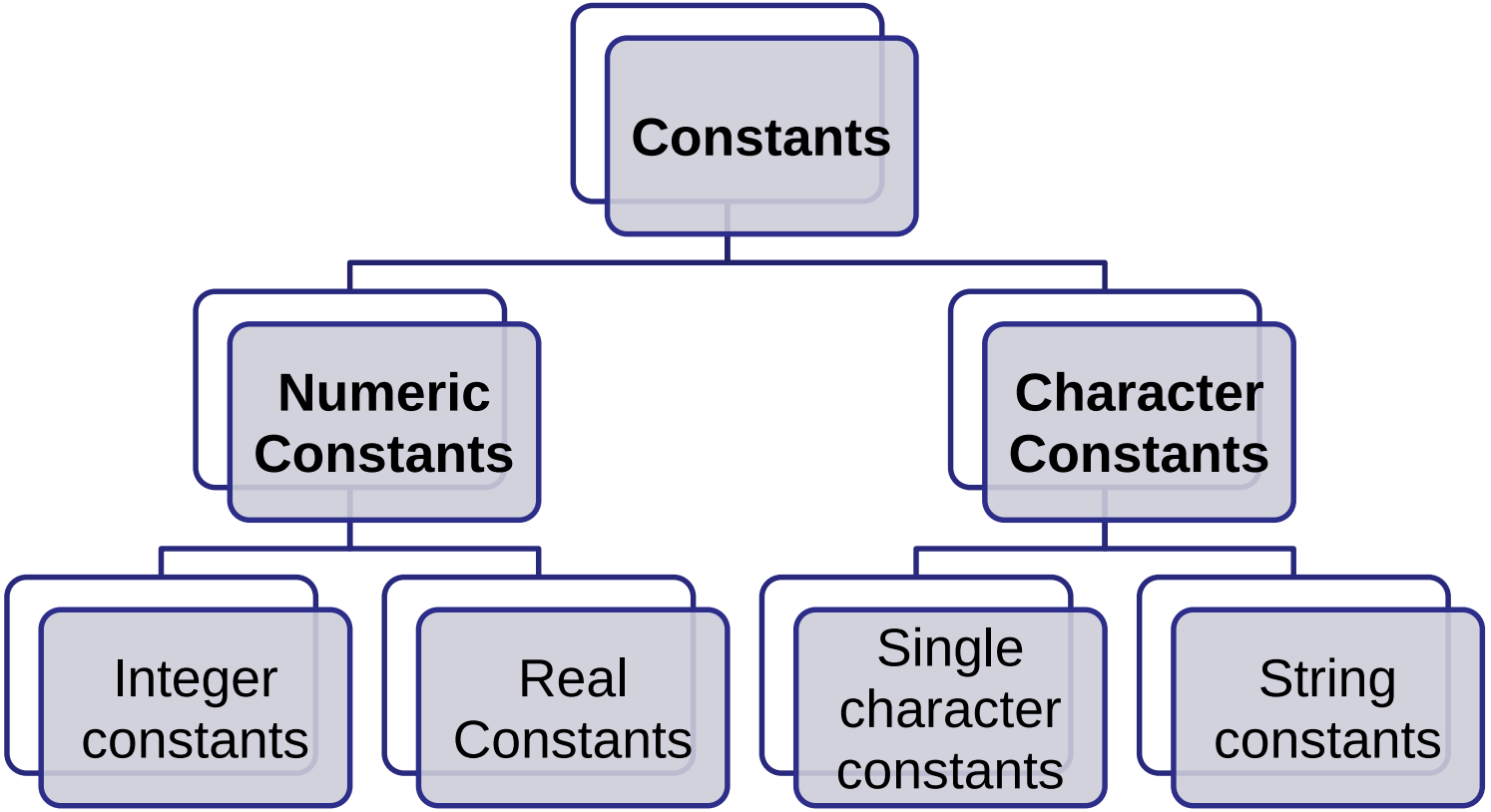
Constants

- ✓ 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


Constants
Numeric Constants
Character Constants
Integer constants
Real Constants
Single character constants
String 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;
}
```

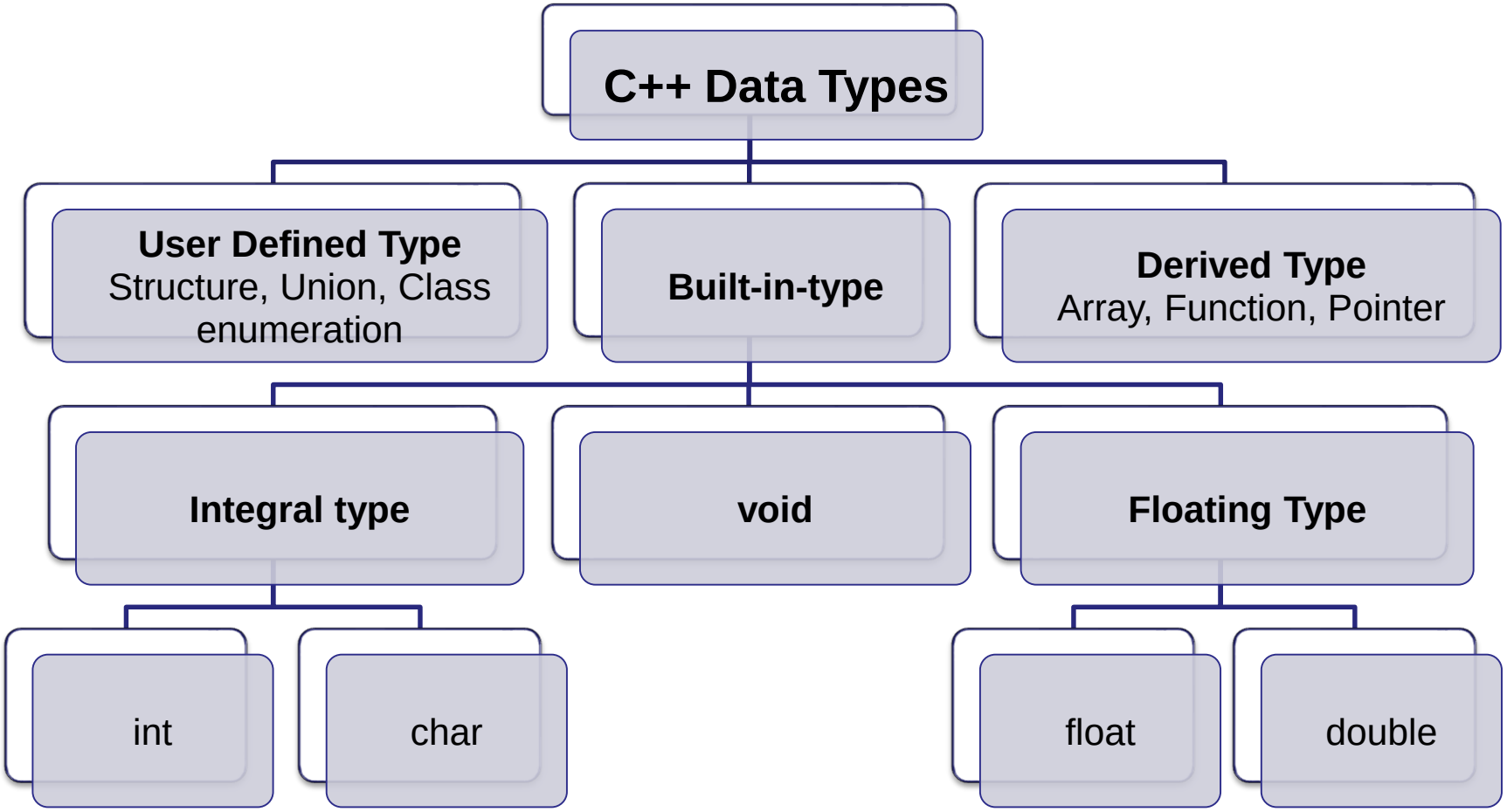
```
#define PI 3.14159
```

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
Structure, Union, Class
enumeration
Built-in-type
Derived Type
Array, Function, Pointer
Integral type
void
Floating Type
int
char
float
double

Data types

C++ Language is rich in its data types

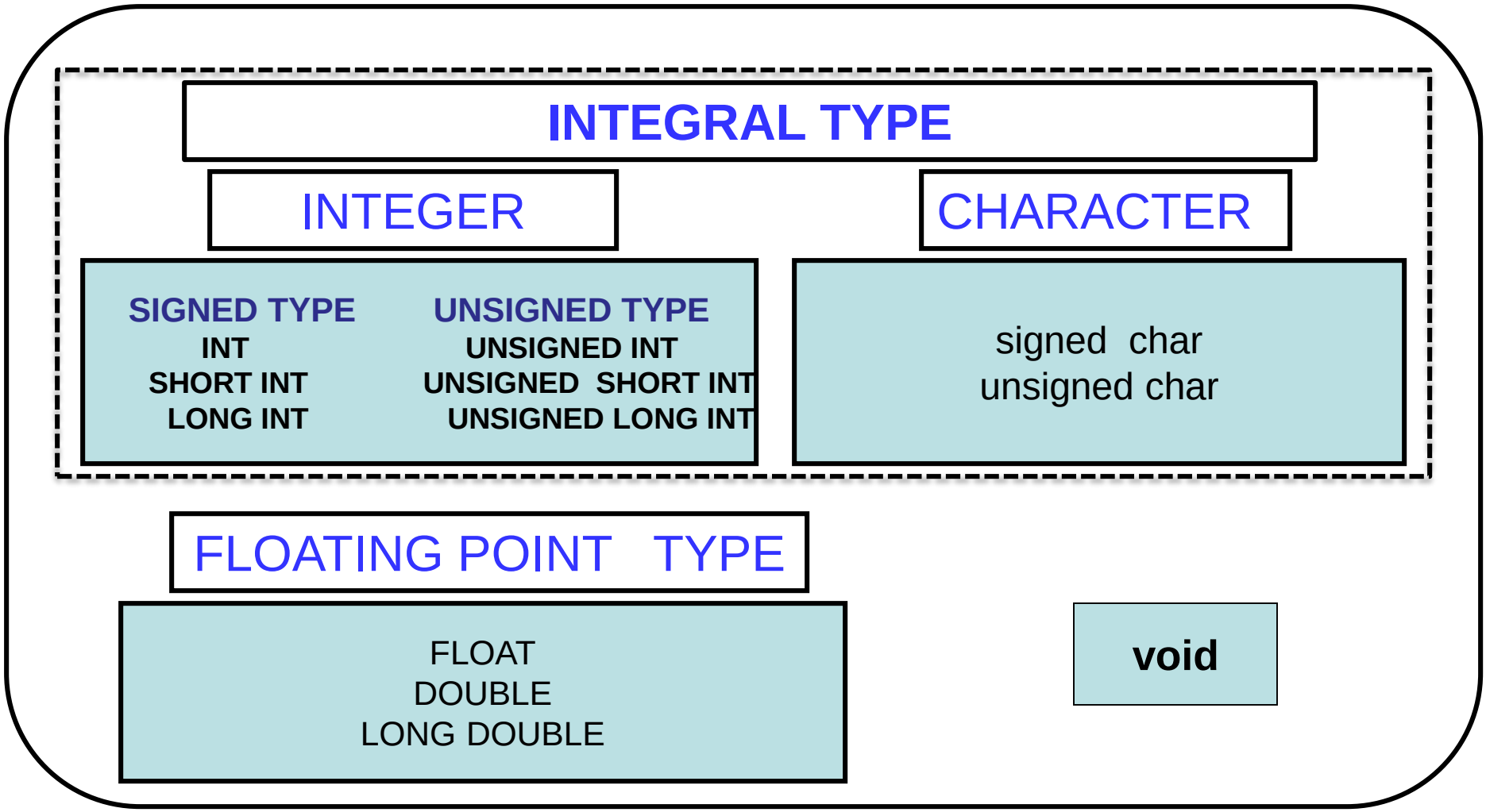
Mainly 4 classes of data types

- 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

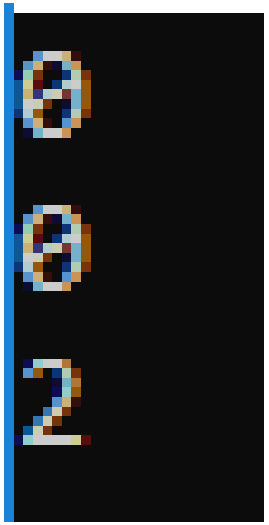
Primary (built-in or Basic) Data types



Example

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

OUTPUT



Example

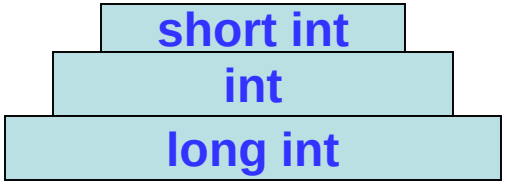
```
 3  int main()
 4  {
 5      int a = 10, b = 20, i = 5;
 6      float f1, f2, f3;
 7
 8      f1 = a / (float)b;
 9      f2 = i / 2;
10      f3 = i / 2.0;
11      cout<<f1<<"\n"<<f2<<"\n"<<f3;
12      return 0;
13  }
```

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**



- 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		

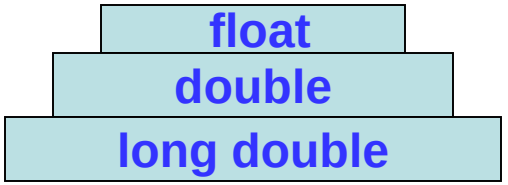
Character Types

- 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
    - ✓ **`'A'` encoded as 65**
    - ✓ **`'a'` encoded as 97**
    - `'+'` encoded as 43
    - **`'Z'` encoded as 90**
    - **`'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`**



SIZE AND RANGE OF VALUES FOR 16-BIT MACHINE (FLOATING POINT)

	Type	Size	Range
Single Precision	float	32 bits 4 bytes	Numbers between 3.4 E-38 and 3.4E+38
Double Precision	double	64 bits 8 bytes	Numbers between 1.7E-308 and 1.7E+308
Long Double Precision	long double	80 bits 10 bytes	Numbers between 3.4E-4932 and1.1E+4932

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

	Type	Size	Range
Single Precision	float		
Double Precision	double		
Long Double Precision	long double		

void

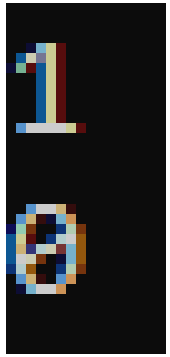
- 2 uses of void are
  - To specify the return type of a function when it is not returning any value
  - To indicate an empty argument list to a function

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**

```
3   int main()
4   {
5       bool y = true;
6       bool n = false;
7       cout<<y<<"\n"<<n;
8       return 0;
9   }
```

OUTPUT



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* $V_1, V_2, \ldots, V_n$

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

```
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;
}
```

Review of Data Types

- ✓ What is a data type?
- ✓ List the different category of data tyes?
- ✓ Name the types under Built-in Data types.
- ✓ Discuss the variations of the following:

  int, char, float
- ✓ What is void? Why it is required?
- ✓ How you can declare variables?
- ✓ What is initialization? Why it is required?
- ✓ List the six different steps involved in executing a program.

Program to compute sum of digits of a 4-digit number

```
#include<iostream>
using namespace std;
int main()
{
int num, digit, sum=0;
cout<<“enter 4 digit no.”;
cin>>num;
digit= num%10;
num=num/10;
sum=sum + digit;
digit= num%10;
num=num/10;
sum=sum + digit;
digit= num%10;
num=num/10;
sum=sum + digit + num;
cout<<“\n”<<“sum of digits=“<<sum;
return 0;
}
```

Try the Following problems

1. Write a C++ program to read the price of an item in decimal form(like 15.95) and print the output in paise(like 1595 paise).
2. Write a C++ program to covert distance in mm to cm, inch, feet (1 cm =10mm, 1inch=2.5cm, 1 feet =12 inches).

User defined Type declarations

- ***typedef***
  - *Type definition - lets you define your own identifiers.*
- ***enum***
  - *Enumerated data type - a type with restricted set of values.*

User defined Type Declaration

- ***typedef*** **:** general declaration format

  typedef **type identifier;**

  The "type" refers to an existing data type and "identifier" refers to the new name given to the data type.

  After the declaration as follows:

  typedef **int marks;**

  typedef **float units;**

  we can use these to declare variables as shown

  marks *m1,m2[10];* *//m1 & m2[10]* are declared as integer variables

  units *u1, u2;* *//u1 & u2* are declared as floating point variables

  The main advantage of typedef is that we can create meaningful data type names for increasing the readability of the program.

User defined Type Declaration

- ***enum*** **data type :** general declaration format

  enum identifier {value1, value2,..,value$_n$};

  The “identifier” is a user defined enumerated data type which can be used to declare variables that can have one of the values known as *enumeration constants.*

  After this declaration as follows:

  enum identifier v1,v2,…v$_n$;

  enumerated variables v1,v2,..,v$_n$ can only have one of the values value1, value2,…,value$_n$

User defined Type Declaration

E.g.:

```
enum  day {Monday, Tuesday,… ,Sunday} ;
enum  day week_st, week_end;
      week_st = Monday; week_end= Friday;
      if(week_st == Tuesday)
            week_end = Saturday;
```

Compiler automatically assigns integer starting with 0 to all enumeration constants. But can be overridden.

E.g.: if you give

```
enum  day { Monday=1, Tuesday,…, Sunday};
```

Monday is assigned 1 & subsequent constants incremented by one.

enum – example

If declared; ***enum*** **day { Monday=1, Tuesday,…, Sunday};**

```
cout<<"\n\nEnter  n >1 &<7., 0 for Exit:- ";
  cin>>i;
  switch(i)
  {
            case Monday:
                        cout<<“Monday.";
                        break;
            case Tuesday
                        cout<<“ Tuesday.";
                        break;
            case Wednesday:
                        cout<<“ Wednesday.";
                        break;
            case Thursday:
                        cout<<“ Thursday.";
                        break;
            case Friday:
                        cout<<“ Friday.";
                        break;
            case Saturday:
                        cout<<“\Saturday.";
                        break;
            case Sunday:
                        cout<<“Sunday.";
                        break;
            case 0:
                 exit(0);
            default:
            cout<<"\nInvalid Entry. Enter  0-7.";
                        break;
}
```