Data Type


Pemrograman
C++
Dasar
PHP
C++
Ruby
Java
C
C++
Pascal
Program
C++
Python
Python
PL-SQL

Data Transformation

- Data is the representation of information in a manner suitable for communication or analysis by humans or machines
- Data is stored in the memory as a string of binary digits (0 and 1) having finite length. In a machine instruction, a memory location is identified by its address.
- Data can be read from a memory location and a memory location can also be updated
- Programs transform data from one form to another
  - Input data → Output data
  - Stimulus → Response
- Programming languages store and process data in various ways depending on the type of the data; consequently, all data read, processed, or written by a program must have a type
- Two distinguishing characteristics of a programming language are the data types it supports and the operations on those data types

Data Types

- Data types (or sometimes just called ‘types’)are the descriptions of the containers of the data
- The basic idea of typing data is to give some useful meaning to what is ultimately just binary digits.
- Describing this data using types makes it easier to understand, manipulate or retrieve.
- Data is held within the computers RAM until needed by the program.
- The amount of RAM needed to hold each data type depends on what type of data type is being used

A Data Type

- A data type is
  - A set of values AND
  - A set of operations on those values
- A data type is used to
  - Identify the type of a variable when the variable is declared
  - Identify the type of the return value of a function
  - Identify the type of a parameter expected by a function

A Data Type

- When the compiler encounters a declaration for a variable, it sets up a memory location for it
- An operator used on a variable or variables is legal only if
  - The operator is defined in that programming language for a variable of that type
  - The variable or variables involved with the operator are of the same or compatible types

Rules for Constructing Identifiers in C

- Capital letters A-Z, lowercase letters a-z, digits 0-9, and the underscore character
- First character must be a letter or underscore
- Usually only the first 32 characters are significant
- There can be no embedded blanks
- Keywords cannot be used as identifiers
- Identifiers are case sensitive

Identifiers refer to the names of data types, constants, variables, and functions

Classifications of Data Types

C programming language which has the ability to divide the data into different types. The type of a variable determine the what kind of values it may take on. The various data types are

- Fundamental Data type

  → Integer, Float, Void, Char, enum

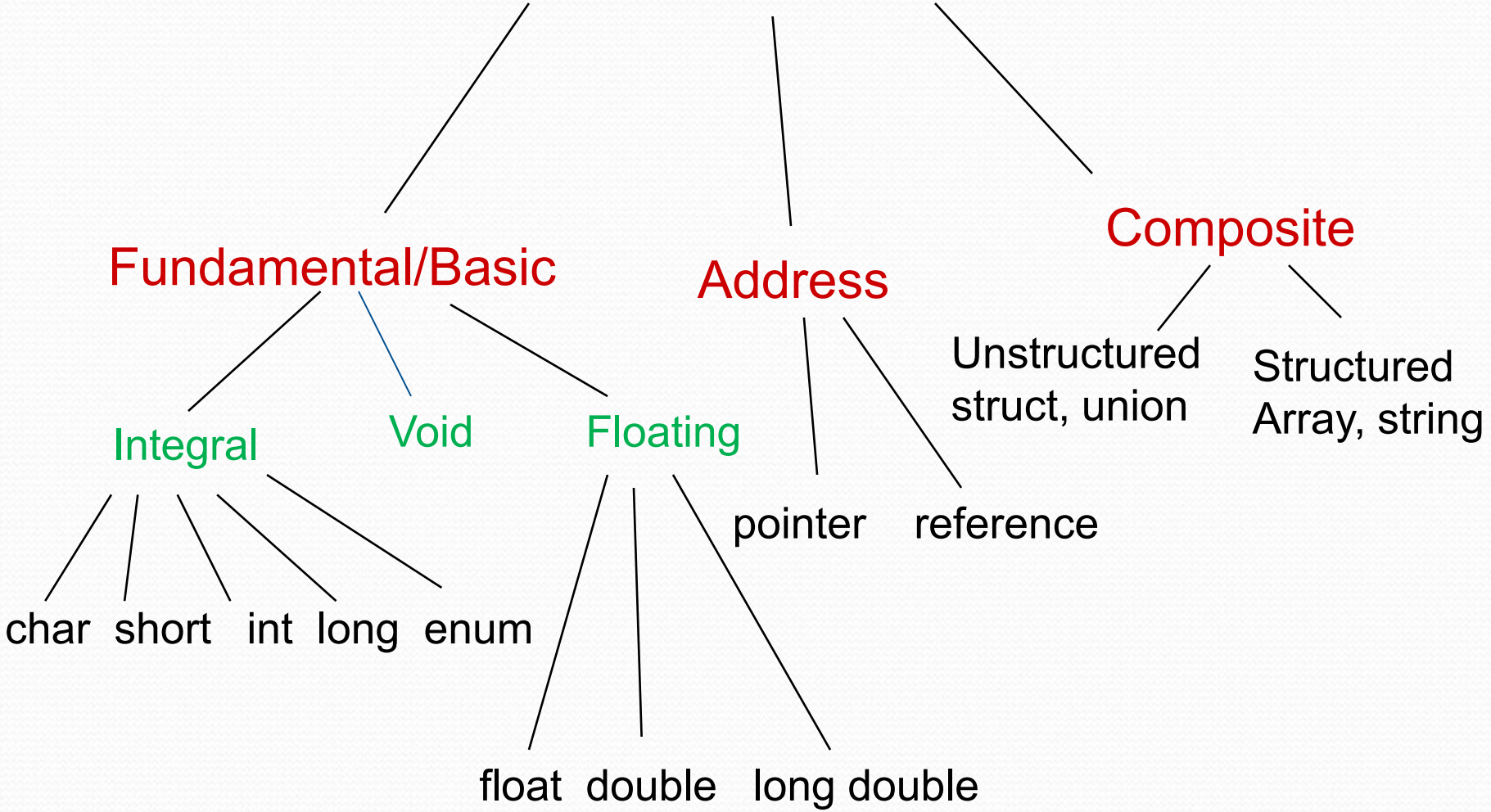
- Address

  →pointer

- Composite

  →struct, union, array, strings

Built-In Data Types



Fundamental Data Types

- **void** – used to denote the type with no values
- **int** – used to denote an integer type
- **char** – used to denote a character type
- **float, double** – used to denote a floating point type
- **enum** -

Composite Data Types

- **Structure** – a collection of related variables of the same and/or different data types. The structure is called a record and the variables in the record are called members or fields
- **Union**
- **Array** – a finite sequence (or table) of variables of the same data type
- **String** – an array of character variables

Fundamental Data Types

- Integral Types
  - Integers are stored in various sizes. They can be signed or unsigned.
  - **Example**
    Suppose an char integer is represented by a byte (8 bits).

    Leftmost bit is sign bit. If the sign bit is 0, the number is treated as positive.

    The largest positive number is 01111111 = $2^7 - 1 = 127$.
    Bit pattern 01001011 = 75 (decimal).
    Negative numbers are stored as two's complement or as one's complement.
    -75 = 10110100 (one's complement).
    -75 = 10110101 (two's complement).

Basic Data Types

- Integral Types
  - **char** — Stored as 8 bits. Unsigned 0 to 255. Signed -128 to 127.
  - **short int** — Stored as 16 bits.
  - **int** — Stored as 16 bits. Unsigned 0 to 65535

    Signed -32768 to 32767 (-2^{15} to $+2^{15}-1$).
  - **long int** — Stored as 32 bits. Unsigned 0 to 4294967295. Signed -2147483648 to 2147483647

Integer Types in C

Type	Range in Typical Microprocessor Implementation
short	-32767 ~ 32767
unsigned short	0 ~ 65535
int	-32767 ~ 32767
unsigned	0 ~ 65535
long	-2147483647 ~ 2147483647
unsigned long	0 ~ 4294967295

Reading and Writing Integers

```
unsigned int u;
scanf("%u", &u);     /* reads u in base 10 */
printf("%u", u);      /* writes u in base 10 */
scanf("%o", &u);      /* reads u in base 8 */
printf("%o", u);      /* writes u in base 8 */
scanf("%x", &u);      /* reads u in base 16 */
printf("%x", u);      /* writes u in base 16*/

short int x;
scanf("%hd", &x);
printf("%hd", x);

long int x;
scanf("%ld", &x);
printf("%ld", x);
```

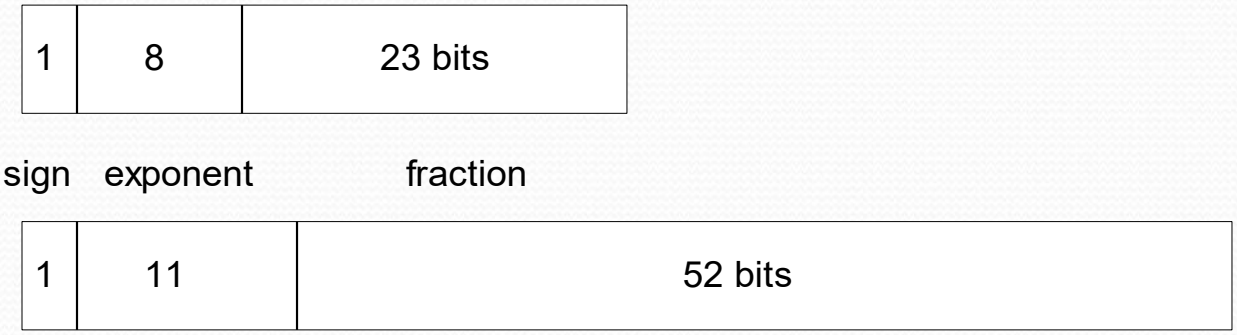
Fundamental Data Types

- Floating Point Numbers
  - Floating point numbers are rational numbers. Always signed numbers.
  - **float** Approximate precision of 6 decimal digits .
    - Typically stored in 4 bytes with 24 bits of signed mantissa and 8 bits of signed exponent.
  - **double** Approximate precision of 14 decimal digits.
    - Typically stored in 8 bytes with 56 bits of signed mantissa and 8 bits of signed exponent.
  - **Long** Approximate precision of 18 decimal digits
    - Typically stored in 10 bytes with 72 bits of signed mantissa and 8 bits of signed exponent.

Floating-point

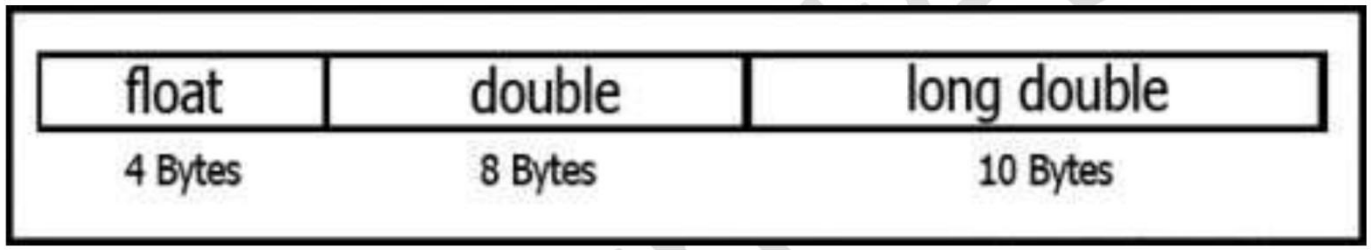
- Historically, floating-points have been stored in a variety of formats
- Same basic components: sign, fraction/mantissa, exponent
- Example : **4.0 = 0010 * 2^{0001} = 2 * 2^{1}**
- In 1985, IEEE floating-point formats were standardized

(sign) fraction x $2^{exponent}$



Floating-Point Types in C

Type	Approximate Range*	Significant Digits*
Float(32 bit)	10^{-37} ~10^{38}	6
Double(64 bit)	10^{-307} ~10^{308}	15
long double(80 bit)	10^{-4931} ~10^{4932}	19



Floating Types

float	single_precision_floating_point
double	double_precision_ floating_point
long double	extended_precision_ floating_point

```
double x;
scanf("%lf", &x);
printf("%lf", x);
```

```
long double x;
scanf("%Lf", &x);
printf("%Lf", x);
```

Type double Constants

Valid double Constants	Invalid double Constants
3.14159	150 (no decimal point)
0.005	.12345e(missing exponent)
12345.0	15e-0.3(0.3 is invalid)
15.0e-04 (0.0015)	
2.345e2 (234.5)	12.5e.3(.3 is invalid)
1.15e-3 (0.00115)	34,500.99(, is not allowed)
12e+5 (1200000.0)	

Floating Point Arithmetic

- Representation
  - All floating point numbers are stored as

$$\pm 0.d_1 d_2 \cdots d_p \times B^e$$

  - such that d_1 is nonzero. B is the base. p is the precision or number of significant digits. e is the exponent. All these put together have finite number of bits (usually 32 or 64 bits) of storage.
  - Example
  - Assume B = 10 and p = 3.
  - 23.7 = +0.237E2
  - 23.74 = +0.237E2
  - 37000 = +0.370E5
  - 37028 = +0.370E5
  - -0.000124 = -0.124E-4

Floating Point Arithmetic

- Representation
  - $S_k = \{ x \mid B^{k-1} <= x < B^k \}$. Number of elements in each S_k is same. In the previous example it is 900.
  - Gap between seuccessive numbers of S_k is B^{k-p}.
  - B^{1-p} is called machine epsilon. It is the gap between 1 and next representable number.
  - Underflow and Overflow occur when number cannot be represented because it is too small or too big.
  - Two floating points are added by aligning decimal points.
  - Floating point arithmetic is not associative and distributive.

Character Types

- Data Type char (8 bit)
- Rrepresent an individual character value
- include a letter, a digit, a special symbol
- Example ‘A’ ‘z’ ‘2’ ‘9’ ‘ * ’ ‘ : ’ ‘ ” ’ ‘ ‘
- ‘a‘, ‘\t’, ‘\n’, ‘\0’, etc. are character constants
- There are signed and unsigned chars; both occupy 1 byte each, but having different ranges.
- Unsigned characters have values between 0 and 255,
- signed characters have values from –128 to 127.

Character Types

```
char ch;
int i;
i = ‘a’;                    /* i is now 97 */
ch = 65;                    /* ch is now ‘A’ */
ch = ch + 1;                /* ch is now ‘B’ */
ch++;                       /* ch is now ‘C’ */

if(‘a’ <= ch && ch <= ‘z’)

for(ch = ‘A’; ch <= ‘Z’; ch++)
```

Void

- The void type has no values therefore we cannot declare it as variable as we did in case of integer and float.
- The void data type is usually used with function to specify its type.

enum - Enumeration Constants

- Enum is another user-defined type consisting of a set of named constants called enumerators.
- Using a keyword enum, it is a set of integer constants represented by identifiers.
- As said before, by default, the first enumerator has a value of 0, and each successive enumerator is one larger than the value of the previous one, unless you explicitly specify a value for a particular enumerator.
- Enumerators needn't have unique values within an enumeration.
- The name of each enumerator is treated as a constant and must be unique within the scope where the enum is defined

Enumeration

- Enumeration is a user-defined data type. It is defined using the keyword enum and the syntax is:

  enum tag_name {name_0, …, name_n} ;

- The tag_name is not used directly. The names in the braces are symbolic constants that take on integer values from zero through n. As an example, the statement:

  enum colors { red, yellow, green } ;

- creates three constants. red is assigned the value 0, yellow is assigned 1 and green is assigned 2.

enum - Enumeration Constants

- The values in an enumstart with 0, unless specified otherwise, and are incremented by 1. For example, the following enumeration,

  enum days {Mon, Tue, Wed, Thu, Fri, Sat, Sun};

- Creates a new data type, enum days, in which the identifiers are set automatically to the integers 0 to 6.
- To number the days 1 to 7, use the following enumeration,

  enum days {Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun};

- Or we can re-arrange the order,

  enum days {Mon, Tue, Wed, Thu = 7, Fri, Sat, Sun};

Questions?