# Basic Data Types

Dr. Nguyen Hua Phung

HCMC University of Technology, Viet Nam

01, 2015

# Outline

# Introduction

- A data type defines
  - a collection of data objects
  - a set of predefined operations
- Abstract Data Type: Interface (visible) are separated from the representation and operation (hidden)
- Uses of type system:
  - Error detection
  - Program modularization assistant
  - Documentation
- Think of variables as descriptors

# Integer

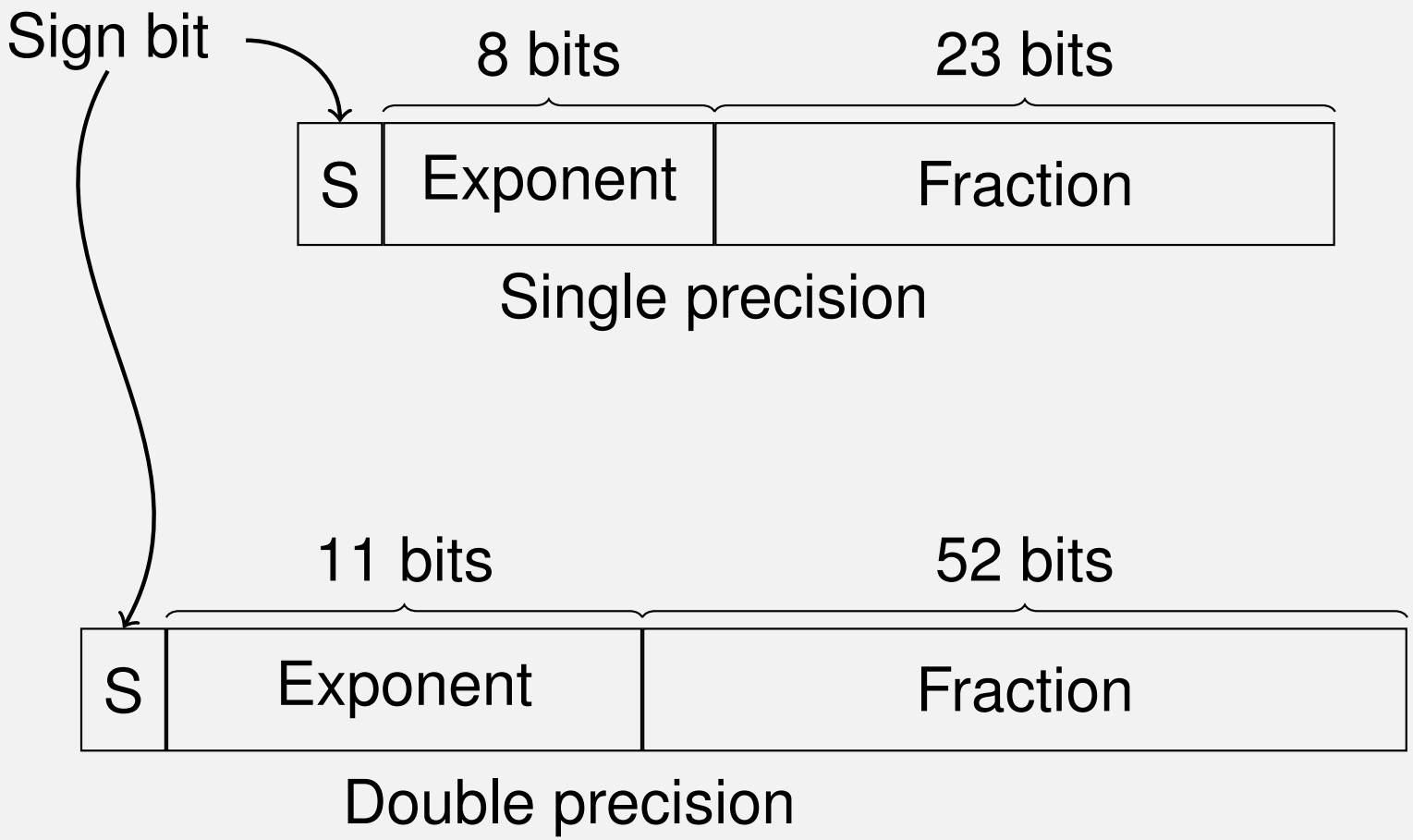
- Languages may support several sizes of integer
  - C++'s signed integer sizes: *char*, *short*, *int*, *long*
- C++ language includes *signed*/*unsigned* integers
- Supported directly by hardware: a string of bits
- To represent negative numbers: two's complement
- The bit-width is an unspecified system-dependent unit.

| Integer type | Minimum Size | Relative Size |
|---|---|---|
| *char* | 8 bits | one unit |
| *short* | 16 bits | at least as wide as *char* |
| *int* | 16 bits | at least as wide as *short* |
| *long* | 32 bits | at least as wide as *int* |

# Floating-Point

- Model real numbers, but only as approximations
- C++:
  - *float*
  - *double*
  - *long double*
- Precision and range
- IEEE Floating-Point Standard 754

# IEEE-754


Sign bit
8 bits
23 bits
S
Exponent
Fraction
Single precision
11 bits
52 bits
S
Exponent
Fraction
Double precision

# Bool

- Simplest of all
- Range of values: two elements, one for “true” and one for “false”
- Could be implemented as bits, but often as bytes
- Example,
  *bool isLoop;*

# Character Literals

- a constant that is composed of a character surrounded by single quotation marks.
- Stored as numeric codings
- Narrow-character: *char*, *'a'*
- Wide-character:*wchar_t*, *L'a'*
- Read more about Escape Sequences, Unicode Characters

# Enumeration Types

- All possible values, which are named constants, are provided in the definition
- ***enum*** *days {Mon, Tue, Wed, Thu, Fri, Sat, Sun};*
  *days myDay = Mon, yourDay = Tue;*
- Note:
  - No duplicate named constant
  - Can be coerced to integer
    *int numDay = myDay;*
  - Required to static cast from *int* to an enum type
    *myDay = (days) numday;*
  - No input/output

# Structured Type

- Many *components* which can be accessed individually
- component's type: the *same* (homogeneous) or *different* (heterogeneous)
- the number of components: *fixed* or *changed*
- *operations* on *structured-type object* or *its components*
- Component *insertion/removal* operations
- *creation/destruction* operations

# Array Types

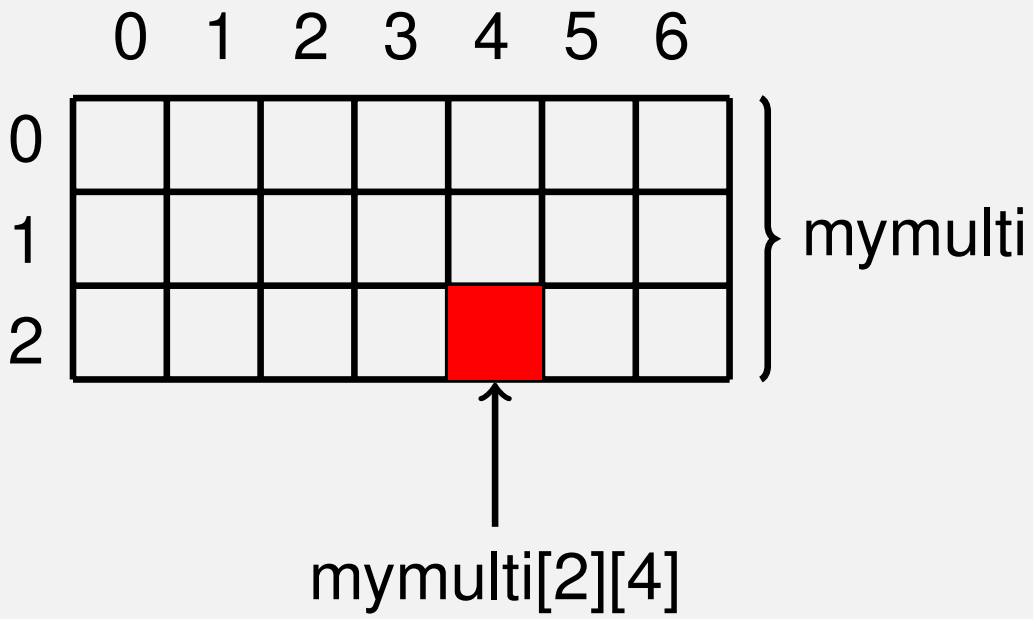
- Collection of *homogeneous* data elements
- Each element is identified by its position relative to the first element and referenced using *subscript expression*
  *type array_name [elements];*
  *int myarr[5];*
- Array indices start from 0:
  *cout « myarr[0];*

# Array Initialization

- C++ allows initialization at the time of storage allocation
  - Arrays of integer:
    *int list [] = {4, 5, 7, 83}*
  - Arrays of character:
    *char name [] = "freddie";*
    *char othername [] = {'a', 'b'};*
  - Arrays of strings
    *char *names [] = {"Bob", "Jake", "Joe"};*

# Multidimension Arrays

- C++: arrays of array
  *mymulti[3][7];*



# String Types

- String literals: sequence of characters enclosed in double quote marks ⇒ "example"
- Use std::string class ⇒ *std::string mystring;*
- Input a string ⇒ getline(cin, mystring, '\n');
- Typical operations
  - Assignment: *mystring = "example";*
  - Comparison (==, >, etc.)
  - Concatenation: mystring +" 1" ⇒ "example 1"
  - Substring reference ⇒ mystring.substr(2,3)

# Record Types

- A record:
  - heterogeneous aggregate of data elements
  - individual elements are identified by names
- C++: *struct*

```
struct type_name {
    member_type1 member1;
    member_type2 member2;
    member_type3 member3;
} object_name;
```

# Example of struct type

```
struct scv {
    std::string family_name;
    std::string first_name;
    int age;
} svname[30];
svname[0].family_name = "Nguyen";
svname[0].first_name = "Van Teo";
svname[0].age = 23;
svname[1].family_name = "Tran";
svname[1].first_name = "Van Ty";
svname[1].age = 24;
```

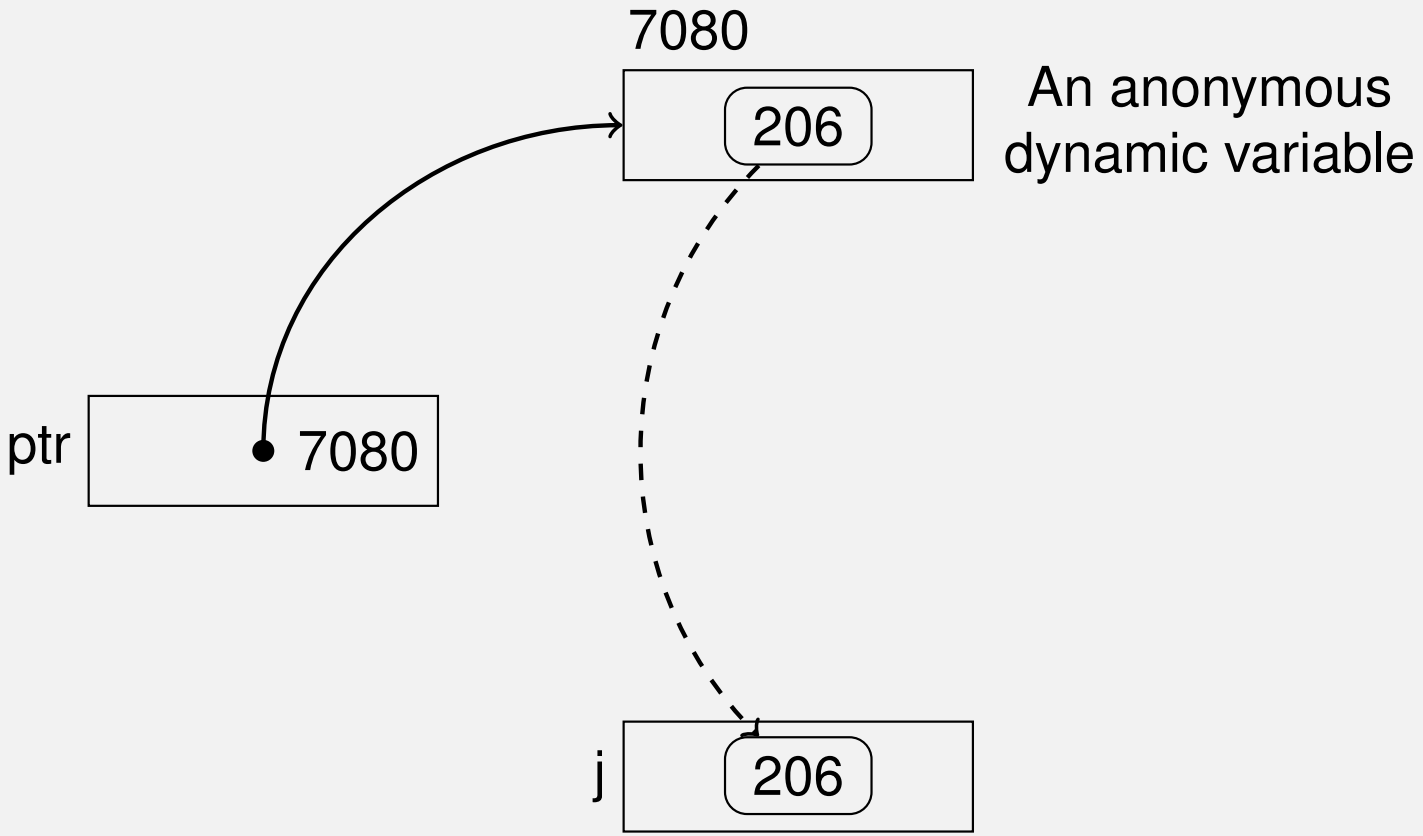
# Pointer and Reference Types

- A *pointer* type variable has a range of values that consists of memory addresses and a special value, *nil*
- Provide the power of indirect addressing
- Provide a way to manage dynamic memory
  - A pointer can be used to access a location in the area where storage is dynamically created (usually called a *heap*)

# Pointer Operations

- Two fundamental operations: assignment and dereferencing
- Assignment is used to set a pointer variable's value to some useful address
- Dereferencing yields the value stored at the location represented by the pointer's value
  - Dereferencing can be explicit or implicit
  - C++ uses an explicit operation via *
    j = *ptr
    sets j to the value located at ptr

# Pointer Assignment Illustrated




The assignment operation j = *ptr

# Pointer Operations

- Pointer points to a record in C/C++
  - Explicit: (*p).name
  - Implicit: p -> name
- Management of heap use explicit allocation
  - C: function **malloc**
  - C++: **new** and **delete** operators

# Problems with Pointers

- Dangling pointers (dangerous)
  - A pointer points to a heap-dynamic variable that has been de-allocated
- Lost heap-dynamic variable
  - An allocated heap-dynamic variable that is no longer accessible to the user program (often called *garbage*)

# Pointers in C and C++

```
int *ptr;
int count, init;
...
ptr = &init;
count = *ptr;
```

- Extremely flexible but must be used with care
- Pointers can point at any variable regardless of when it was allocated
- Used for dynamic storage management and addressing

# Pointers in C and C++

- Pointer arithmetic is possible

```
int list [10]; int *ptr; ptr = list;
*(ptr + 1)
*(ptr + index)
*ptr[index]
```

- Explicit dereferencing and address-of operators
- Domain type need not be fixed (void *)
- void * can point to any type and can be type checked (cannot be de-referenced)

# Reference Types

- Pointers refer to an address, references refer to object or value
- C++ includes a special kind of pointer type called a reference type that is used primarily for formal parameters

```
int A;
int &rA = A;
A = 1;
cout << rA << endl;
rA++;
cout << A << endl
```

# Relationship to Pointers

| Reference Type | Pointer |
|---|---|
| `int A;`<br>`int& rA = A;` | `int A;`<br>`int* pA = &A;` |
| rA $\Rightarrow$ A | *pA $\Rightarrow$ A |
| N/A | pA++ |
| cannot reseated | pA = &B |
| cannot be null | pA = null |
| cannot be uninitialized | int* pA |

# Summary

- Type system is mainly used to error detection
- Primitive type
- Structured type