C++ Programming Essentials

Unit 3: Basic Abstract Data Types

CHAPTER 9: SIMPLE DATA TYPES: BUILT-IN AND USER-DEFINED

DR. ERIC CHOU IEEE SENIOR MEMBER

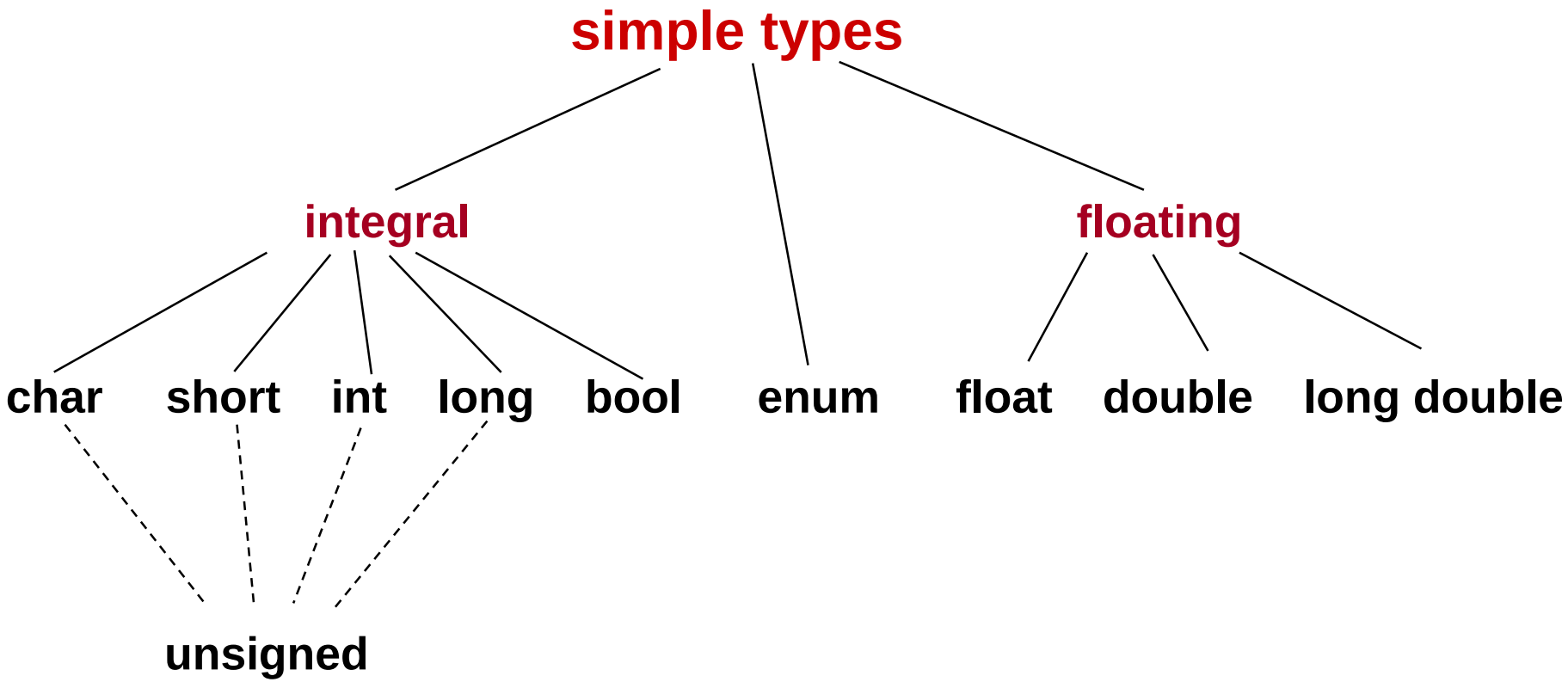
LECTURE 1

Simple Data Types

Chapter 9 Topics

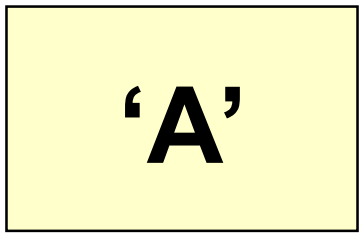
- External and Internal Representations of Data
- Integral and Floating Point Data Types
- Using Combined Assignment Operators
- Prefix and Postfix Forms of Increment and Decrement Operators
- Using Ternary Operator
- Using Type Cast Operator
- Using an Enumeration Type
- Creating and Including User-Written Header Files

C++ Simple Data Types



By definition,

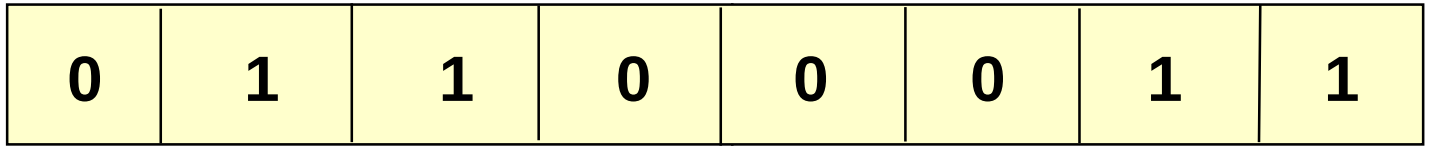
- the size of a C++ char value is always 1 byte.



exactly one byte of memory space

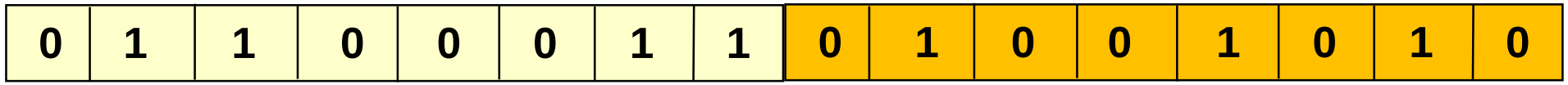
- Sizes of other data type values in C++ are machine-dependent.

Using one byte (= 8 bits),



- HOW MANY DIFFERENT NUMBERS CAN BE REPRESENTED USING 0's and 1's?
- Each bit can hold either a 0 or a 1. So there are just two choices for each bit, and there are 8 bits.
- $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^8 = 256$

Similarly, using two bytes (= 16 bits),



$$2^{16} = 65,536$$

- DIFFERENT NUMBERS CAN BE REPRESENTED.
- If we wish to have only one number representing the integer zero, and half of the remaining numbers positive, and half negative, we can obtain the 65,536 numbers in the range below :
- -32,768 . . . . 0 32,767

Some Integral Types

Type	Size in Bytes	Minimum Value	Maximum Value
char	1	-128	127
short	2	-32,768	32,767
int	2	-32,768	32,767
long	4	-2,147,483,648	2,147,483,647

NOTE: Values given for one machine. Actual sizes are machine-dependent.

Data Type `bool`

- domain contains only 2 values, true and false
- allowable operation are the logical (!, &&, ||) and relational operations

Operator `sizeof`

DEFINITION

C++ has a unary operator named sizeof that yields the size on your machine, in bytes, of its single operand. The operand can be a variable name, or it can be the name of a data type enclosed in parentheses.

```
int  age ;

cout   <<  “Size in bytes of variable age is “  <<  sizeof  age   <<  endl ;

cout   <<  “Size in bytes of type float is “  <<  sizeof ( float )   <<  endl ;
```

The only guarantees made by C++ are . . .

- 1 = sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)
- 1 <= sizeof (bool) <= sizeof (long)
- sizeof (float) <= sizeof (double) <= sizeof (long double)
- char is at least 8 bits
- short is at least 16 bits
- long is at least 32 bits

Exponential (Scientific) Notation

2.7E4 means 2.7×10^{4} =

2.7000 = 27000.0

2.7E-4 means 2.7×10^{-4} =

0002.7 = 0.00027

Floating Point Types

Type	Size in Bytes	Minimum Positive Value	Maximum Positive Value
float	4	3.4E-38	3.4E+38
double	8	1.7E-308	1.7E+308
long double	10	3.4E-4932	1.1E+4932

NOTE: Values given for one machine. Actual sizes are machine-dependent.

More about Floating Point Types

- floating point constants in C++ like 94.6 without a suffix are of type double by default
- to obtain another floating point type constant a suffix must be used
- the suffix F or f denotes float type, as in 94.6F
- the suffix L or l denotes long double, as in 94.6L

Header Files `climits` and `cfloat`

- contain constants whose values are the maximum and minimum for your machine
- such constants are FLT_MAX, FLT_MIN, LONG_MAX, LONG_MIN

```
#include  < climits >
using  namespace  std ;
      .
      .
      .
cout  << "Maximum long is "  << LONG_MAX  << endl ;

cout  << "Minimum long is "  << LONG_MIN  << endl ;
```

LECTURE 2

Review of Operators and Assignments

C++ Has Combined Assignment Operators

```
int   age ;
cin >>  age ;
```

Write a statement to add 3 to age.

```
age  =   age + 3 ;
```

Or,

```
age  +=  3 ;
```

Write a statement to subtract 10 from `weight`

```
int   weight ;
cin >>  weight ;
```

```
weight  =  weight  - 10 ;
```

Or,

```
weight  -=  10 ;
```

Write a statement to divide `money` by 5.0

```
float   money ;

cin >>  money ;
```

```
money  =  money / 5.0 ;
```

Or,

```
money  /= 5.0 ;
```

Write a statement to double `profits`

```
float   profits ;

cin >>  profits ;
```

```
profits  =  profits * 2.0 ;
```

Or,

```
profits  *=  2.0 ;
```

Write a statement to raise `cost` 15%

```
float   cost;
cin >>  cost;
cost  =  cost  +  cost * .15 ;
```

Or,

```
cost  = 1.15 * cost;
```

Or,

```
cost  *=  1.15 ;
```

Which form to use?

- when the increment (or decrement) operator is used in a "*stand alone*" statement solely to add one (or subtract one) from a variable's value, it can be used in either prefix or postfix form

BUT…

- when the increment (or decrement) operator is used in a statement with other operators, the prefix and postfix forms can yield *different* results

LET'S SEE HOW. . .

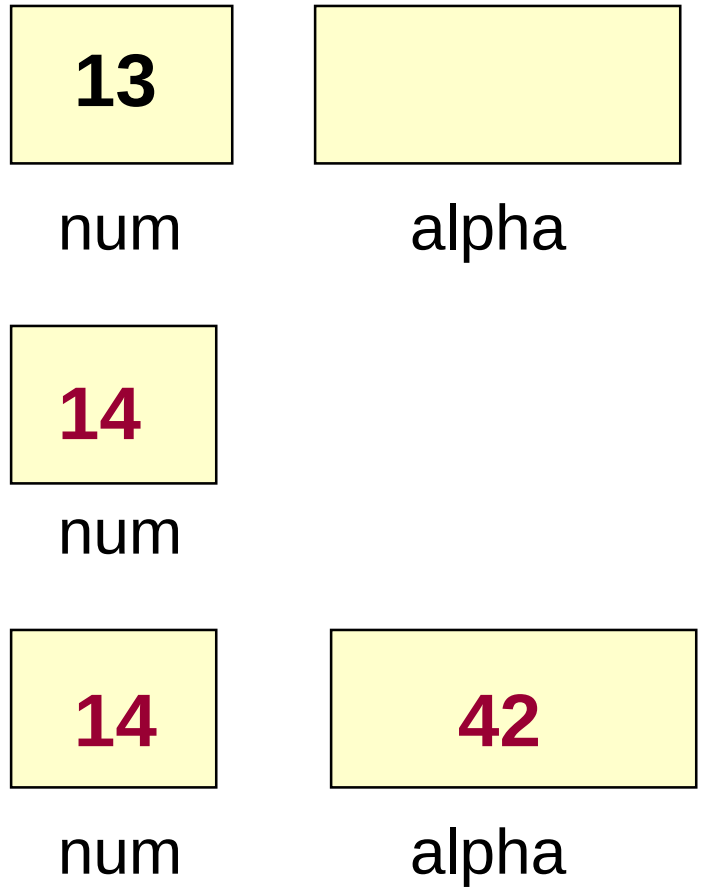
PREFIX FORM

“First increment, then use ”

```
int  alpha ;
int  num ;

num = 13;


alpha = ++num * 3;
```

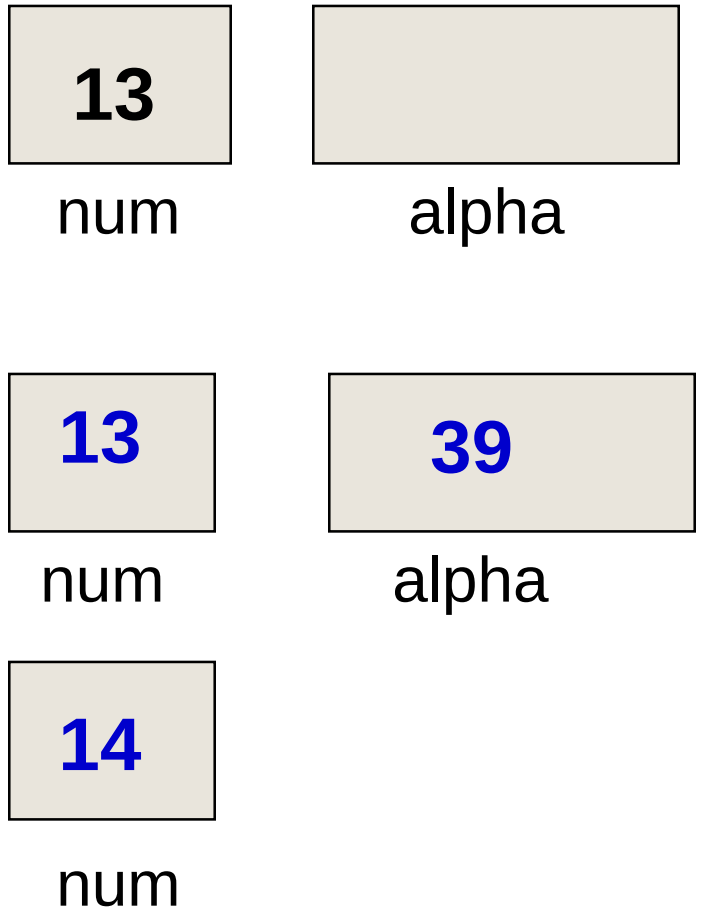


POSTFIX FORM “Use, then increment”

```
int  alpha ;
int  num ;

num = 13;


alpha = num++   * 3;
```



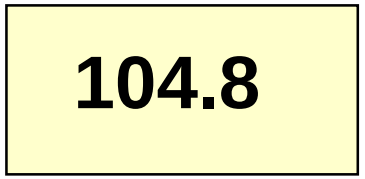
LECTURE 3

Review of Operators

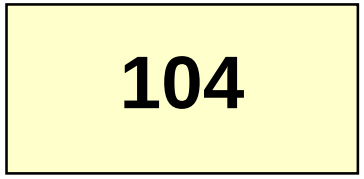
Type Cast Operator

- The C++ cast operator is used to explicitly request a type conversion. The cast operation has two forms.

```
int     intVar;
float   floatVar  =  104.8 ;
intVar  =  int ( floatVar ) ;          // functional notation, OR
intVar  = ( int ) floatVar ;           // prefix notation uses ( )
```



floatVar



intVar

Ternary (three-operand) Operator

condition ? val1 : val0

SYNTAX

```
Expression1 ? Expression2 : Expression3
```

MEANING

If *Expression1* is true (nonzero), then the value of the entire expression is *Expression2*. Otherwise, the value of the entire expression is *Expression 3*.

FOR EXAMPLE . . .

Using Conditional Operator

```
float    Smaller ( float  x,  float  y )
// Finds the smaller of two float values
// Precondition:       x assigned  &&  y assigned
// Postcondition:      Function value  ==   x, if x < y
//                      ==  y, otherwise
{
    float    min;

    min  = ( x < y )  ?   x :  y  ;
    return  min  ;
}
```

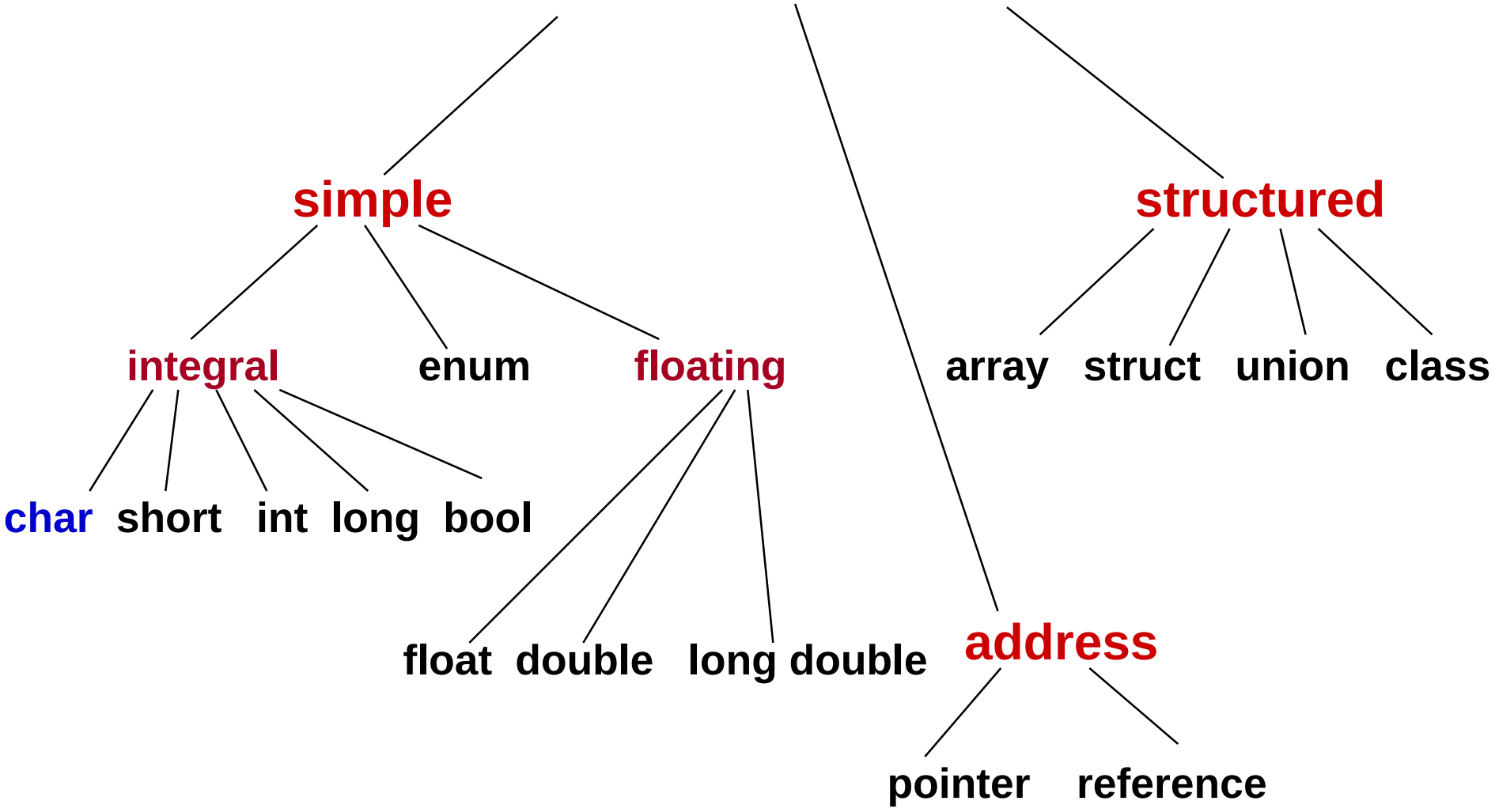
C++ Operator Precedence (highest to lowest)

Operator	*Associativity*
( )	Left to right
unary: ++ -- ! + - (cast) sizeof	Right to left
* / %	Left to right
+ -	Left to right
< <= > >=	Left to right
== !=	Left to right
&&	Left to right
||	Left to right
? :	Right to left
= += -= *= /=	Right to left

LECTURE 4

char Data Type

C++ Data Types


simple
structured
integral
enum
floating
array
struct
union
class
char
short
int
long
bool
float
double
long double
address
pointer
reference

ASCII and EBCDIC

- ASCII (pronounced ask-key) and EBCDIC are the two character sets commonly used to represent characters internally as integers
- ASCII is used on most personal computers, and EBCDIC is used mainly on IBM mainframes
- using ASCII the character 'A' is internally stored as integer 65. Using EBCDIC the 'A' is stored as 193. In both sets, the successive alphabet letters are stored as successive integers. This enables character comparisons with 'A' less than 'B', etc.

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	SoH	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	SoTxt	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	EoTxt	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	EoT	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enq	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Ack	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Bsp	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	HTab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	LFeed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	VTab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	FFeed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	CR	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	SOut	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	SIn	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	DLE	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	DC1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	DC2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	DC3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	DC4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	NAck	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Syn	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	EoTB	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Can	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	EoM	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Sub	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Esc	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	FSep	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	GSep	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	RSep	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	USep	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Delete

Incrementing `char` Variable

- because char variables are stored internally as integers, they can be incremented and compared

EXAMPLE

```
char  ch;
                              // loop to print out letters A thru Z

for  (ch = 'A' ; ch <= 'Z' ;  ch++ )
{
        cout  <<  ch ;
}
```

Control Characters

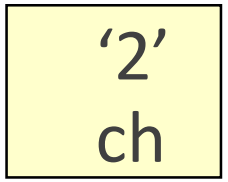
- in addition to the printable characters, character sets also have nonprintable control characters to control the screen, printer, and other hardware
- in C++ programs, control characters are represented by escape sequences. Each escape sequence is formed by a backslash followed by one or more additional characters

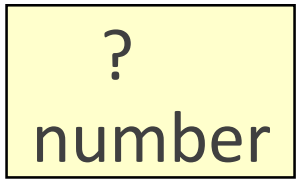
Some Escape Sequences

\n	Newline (Line feed in ASCII)
\t	Horizontal tab
\b	Backspace
\a	Alert (bell or beep)
\\	Backslash
\’	Single quote (apostrophe)
\”	Double quote (quotation mark)
\0	Null character (all zero bits)
\ddd	Octal equivalent (3 octal digits)
\xddd	Hexadecimal equivalent (1 or more hex digits for integer value of character)

Converting `char` digit to `int`

- the successive digit characters ‘0’ through ‘9’ are represented in ASCII by the successive integers 48 through 57 (the situation is similar in EBCDIC)
- as a result, the following expression converts a char digit value to its corresponding integer value






```
char  ch ;
int   number ;
. . .
number = int ( ch - ‘0’ ) ;        // using explicit type cast
```

Character Function Prototypes in < cctype >

```
int   toupper ( int  ch );
// FUNCTION VALUE
//                == uppercase equivalent of ch, if ch is a lowercase letter
//                == ch, otherwise
```

```
int   tolower ( int  ch );
// FUNCTION VALUE
//                == lowercase equivalent of ch, if ch is an uppercase letter
//                == ch, otherwise
```

NOTE: Although parameter and return type are int, in concept these functions operate on character data.

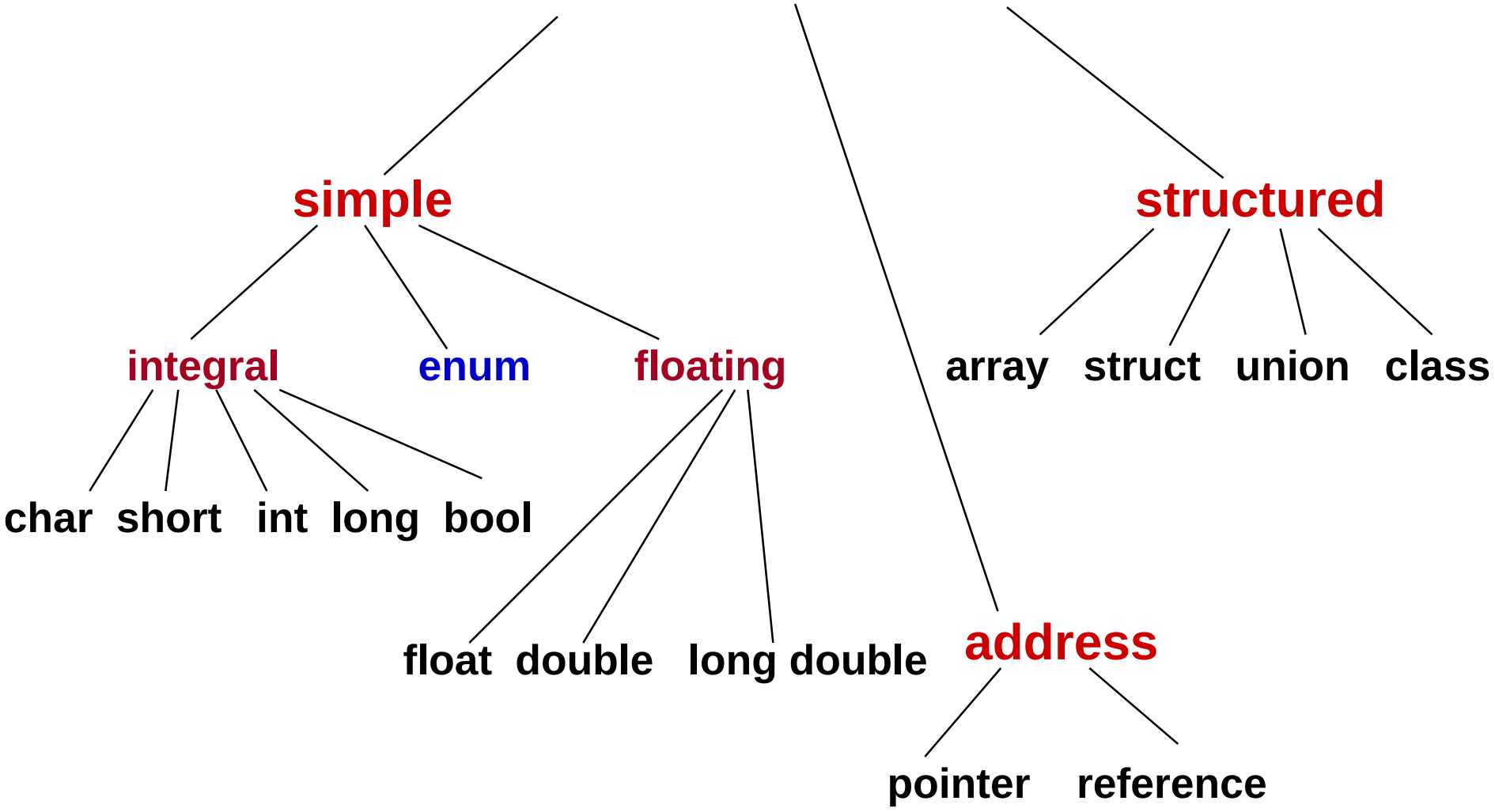
Reading a Yes or No User Response

```
String  inputStr ;
      .
      .
      .
cout  << "Enter Yes or No"  ;
cin  >> inputStr ;
if  ( toupper ( inputStr [0] ) == 'Y' )
{                                   // First letter was 'Y'
      .
      .
      .
}
else if ( toupper ( inputStr [0] ) == 'N' )
{                                   // First letter was 'N'
      .
      .
      .
}
else
      PrintErrorMsg ( ) ;
```

LECTURE 5

enum Data Type

C++ Data Types


simple
structured
integral
enum
floating
array
struct
union
class
char
short
int
long
bool
float
double
long double
address
pointer
reference

typedef statement

- typedef creates an additional name for an already existing data type
- before bool type became part of ISO-ANSI C++, a Boolean type was simulated this way

```
typedef  int  Boolean;
const  Boolean  true = 1 ;
const  Boolean  false  = 0 ;
      .
      .
      .
Boolean dataOK ;
      .
      .
      .
dataOK  =  true ;
```

Enumeration Types

- C++ allows creation of a new simple type by listing (enumerating) all the ordered values in the domain of the type

EXAMPLE

```
enum MonthType { JAN,  FEB,  MAR,  APR,  MAY, JUN,
                 JUL,  AUG,  SEP,  OCT,  NOV,  DEC } ;
```

name of new type

list of all possible values of this new type

enum Type Declaration

```
enum  MonthType { JAN,  FEB,  MAR,  APR,  MAY, JUN,
                  JUL,  AUG,  SEP,  OCT,  NOV,  DEC  } ;
```

- the enum declaration creates a new programmer-defined type and lists all the possible values of that type--any valid C++ identifiers can be used as values
- the listed values are ordered as listed. That is, JAN < FEB < MAR < APR , and so on

you must still declare variables of this type

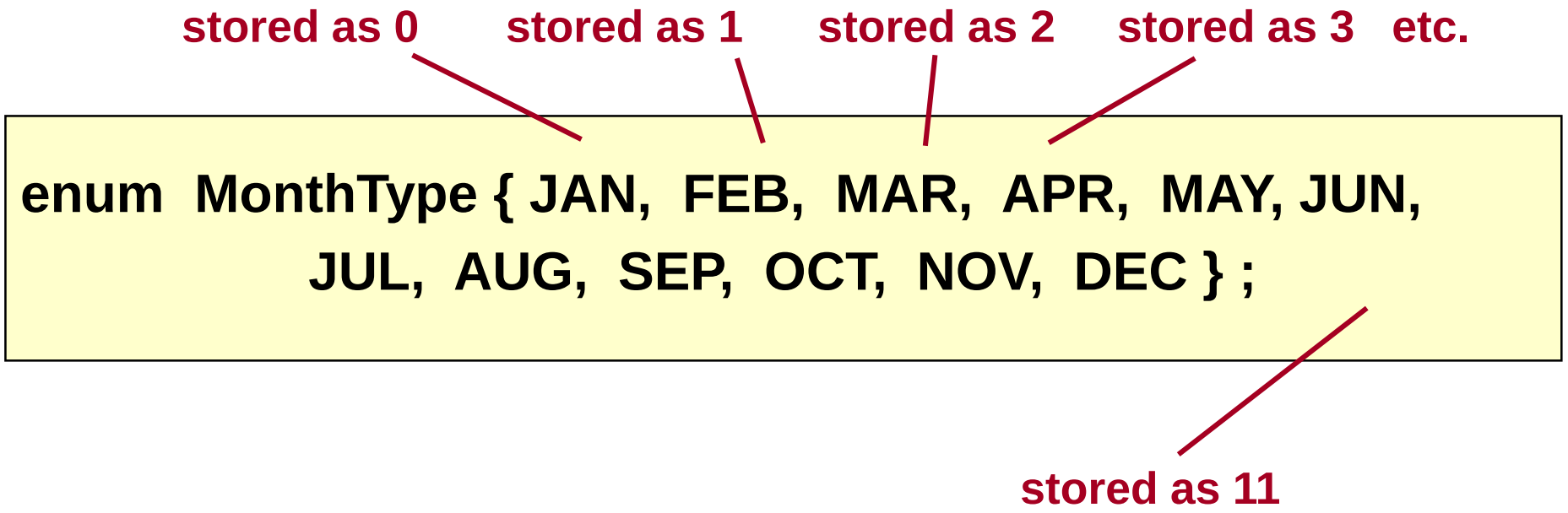
Declaring `enum` Type Variables

```
enum  MonthType { JAN,  FEB,  MAR,  APR,  MAY, JUN,
                  JUL,  AUG,  SEP,  OCT,  NOV,  DEC } ;

MonthType  thisMonth;          // declares 2 variables
MonthType  lastMonth;          // of type MonthType

lastMonth  =  OCT ;          // assigns values
thisMonth  =  NOV ;        // to these variables
     .
     .
     .
lastMonth = thisMonth ;
thisMonth = DEC ;
```

Storage of `enum` Type Variables



Use Type Cast to Increment `enum` Type Variables

```
enum  MonthType { JAN,  FEB,  MAR,  APR,  MAY, JUN,
                  JUL,  AUG,  SEP,  OCT,  NOV,  DEC } ;

MonthType  thisMonth;
MonthType  lastMonth;

lastMonth  =  OCT ;
thisMonth  =  NOV ;
lastMonth = thisMonth ;

thisMonth = thisMonth++ ;          // COMPILE ERROR !

thisMonth = MonthType( thisMonth + 1) ;   // uses type cast
```

More about `enum` Type

- Enumeration type can be used in a Switch statement for the switch expression and the case labels.
- Stream I/O (using the insertion << and extraction >> operators) is not defined for enumeration types. Instead, functions can be written for this purpose.
- Comparison of enum type values is defined using the 6 relational operators (< , <= , > , >= , == , !=).
- An enum type can be the return type of a value-returning function in C++.

SOME EXAMPLES . . .

```
MonthType  thisMonth;
switch ( thisMonth ){              // using enum type switch expression
  case   JAN   :
  case   FEB   :
  case   MAR  :   cout << "Winter quarter" ;
                              break ;
  case   APR  :
  case   MAY  :
  case   JUN   :   cout << "Spring quarter" ;
                              break ;
  case   JUL   :
  case   AUG  :
  case   SEP   :   cout << "Summer quarter" ;
                              break ;
  case   OCT  :
  case   NOV  :
  case   DEC  :   cout << "Fall quarter" ;
}
```

Using `enum` type Control Variable with **for** Loop

```
enum  MonthType { JAN,  FEB,  MAR,  APR,  MAY, JUN,
                  JUL,  AUG,  SEP,  OCT,  NOV,  DEC } ;

void  WriteOutName ( /* in */ MonthType ) ;                    // prototype
            .
            .
            .
MonthType  month ;

for  (month = JAN ;  month <= DEC ;  month = MonthType (month + 1 ) )
{                                          // requires use of type cast to increment

 WriteOutName ( month ) ;    // function call to perform output
            .
            .
            .
}
```

```
void   WriteOutName ( /* in */ MonthType   month )
{switch ( month ) {
          case   JAN  :  cout  <<   " January " ;          break ;
          case   FEB  :  cout  <<   " February " ;         break ;
          case   MAR  :  cout  <<   " March " ;            break ;
          case   APR  :  cout  <<   " April " ;   break ;
          case   MAY  :  cout  <<   " May " ;     break ;
          case   JUN  :  cout  <<   " June " ;           break ;
          case   JUL  :  cout  <<   " July " ;      break ;
          case   AUG  :  cout  <<   " August " ;           break ;
          case   SEP  :  cout  <<   " September " ;  break ;
          case   OCT  :  cout  <<   " October " ;          break ;
          case   NOV  :  cout  <<   " November " ;       break ;
          case   DEC  :  cout  <<   " December " ;    break ;
 }
}
```

Function with `enum` type Return Value

```
enum SchoolType { PRE_SCHOOL, ELEM_SCHOOL,
  MIDDLE_SCHOOL, HIGH_SCHOOL, COLLEGE } ;
 .
 .
 .
SchoolType   GetSchoolData ( void )
// Obtains information from keyboard to determine school level
// Postcondition:    Function value == personal school level
{
 SchoolType  schoolLevel ;
 int                 age ;
 int                 lastGrade ;
 cout  << "Enter age :  " ;                    // prompt for information
 cin  >> age ;
```

```
if  ( age  <   6 )
        schoolLevel = PRE_SCHOOL ;

else
{       cout  <<  "Enter last grade completed in school :  " ;
        cin  >> lastGrade;
        if  ( lastGrade  <  5 )
                schoolLevel  =  ELEM_SCHOOL ;
        else if  ( lastGrade  <  8 )
                schoolLevel  =  MIDDLE_SCHOOL ;
        else if  ( lastGrade  < 12 )
                schoolLevel  =  HIGH_SCHOOL ;
        else
                schoolLevel  =  COLLEGE ;
   }
 return  schoolLevel ;          // return enum type value
}
```

LECTURE 6

C++ Program in Multiple Files

Multifile C++ Programs

- C++ programs often consist of several different files with extensions such as .h and .cpp
- related typedef statements, const values, enum type declarations, and similar items are often placed in user-written header files
- by using the #include preprocessor directive the contents of these header files are inserted into any program file that uses them

Inserting Header Files

```
#include    <iostream>

#include    "school.h"

int   main (   )
{

          .

          .

}
```

```
// iostream
```

```
enum  SchoolType
{  PRE_SCHOOL,
    ELEM_SCHOOL,
    MIDDLE_SCHOOL,
    HIGH_SCHOOL,
    COLLEGE  } ;
```

.

LECTURE 7

Coercion (Implicit Casting)

Implicit type coercion occurs . . .

whenever values of different data types are used in:

1. arithmetic and relational expressions
2. assignment operations
3. parameter passage
4. returning a function value with return
   (from a value-returning function)

TWO RULES APPLY . . .

Promotion (or widening) in C++. . .

- is the conversion of a value from a “lower” type to a “higher” type-- specifically, for mixed type expressions:
- Step 1. Each char, short, bool, or enumeration value is promoted to int. If both operands are now int, the result is an int expression.
- Step 2. If Step 1 leaves a mixed-type expression, the following precedence of types is used (from lowest to highest):

  int, unsigned int, long, unsigned long, float, double, long double

- The value of the operand of “lower” type is promoted to that of the “higher” type. For an arithmetic expression, the result is an expression of that higher type. For a relational expression, the result is always bool (true or false).

Demotion (or narrowing) . . .

- is the conversion of a value from a “higher” type to a “lower” type, and may cause loss of information

FOR EXAMPLE,


```
float  temperature  =  98.6 ;
int    number ;
number  =  temperature ;      // demotion occurs
```