UNIVERSITY of LOUISVILLE
dare to be great

C Chapter 2: Primary Data Types

CECS130
Introduction to Programming Languages
Instructor: Dr. Roman V. Yampolskiy

Memory

- Computer's long term memory
  - (Hard Drive) – Permanent
- Computer's short term memory
  - (Random Access Memory) – Temporary (Only while power is ON)
- RAM is comprised of fixed-size cells
- Each cell has an address
- To simplify things we give names to memory addresses: variables

Common Variable Characteristics

- Name
  - An easy to remember meaningful identifier
- Type
  - Number, Character…
- Value
  - The data value assigned to the memory location
- Address
  - The address assigned to the variable

Examples of Variables

Variable Name	Value	Type	Memory Address
myNumber	99	integer	FFF4
Result	756.21	float	FHH6
Initial	M	character	FHF2

Memory addresses are represented in the hexadecimal numbering system (Base 16)

Variable Naming

- A variable may be defined using
  - any uppercase or lowercase character
  - a numerical digit (0 through 9)
  - the underscore character (_).
  - The first character of the variable may not be a numerical digit or underscore.
  - Variable names are case sensitive.
  - Decimal points, commas, or any other symbols are not allowed.

Meaningful Variable Naming

- A, b, c2, x make poor variable names
- Variable names must be meaningful
- Good names: taxRate, firstInitial, etc.
- Reduce amount of comments necessary
- Make code easier to follow

Keywords

Keywords are reserved and cannot be used as variable names.

Keywords are lower case.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Data Types

- C supports 4 basic data types:
  - char for Character
  - int for integers
  - float for single precision real numbers
  - double for double precision real numbers

Data Types and Sizes

- C also supports two qualifiers that can be applied to some of these types to alter the size of memory storage used by these variables
  - namely short and long

Data Types and Sizes (cont.)

1. ***Integer***
2. Floating point
3. Double Precision
4. Character

- An **integer** value (integer constants) is any positive or negative number without the decimal point.

  5, -10, 26351, -32117
- An integer may be **signed**, i.e. a + or – sign precedes the value to indicate a positive or a negative value respectively (this is the default).

  +5, -21
- An integer may be **unsigned** which means the value is taken as nonnegative.
- The Maximum and minimum limits on integer values depend on the amount of internal storage used by the compiler to represent the number. These values are shown in table below

Storage Area reserved	Minimum *Signed* Integer Value	Maximum *Signed* Integer Value
1 Byte	-128	127
2 Bytes	-32,768	32,767
4 Bytes	-2,147,483,648	2,147,483,647

Storage Area reserved	Minimum *Unsigned* Integer Value	Maximum *Unsigned* Integer Value
1 Byte	0	255
2 Bytes	0	65,535
4 Bytes	0	4,294,967,295

Data Types and Sizes (cont.)

1. **Integer**
2. ***Floating point***
3. ***Double precision***
4. ***Character***

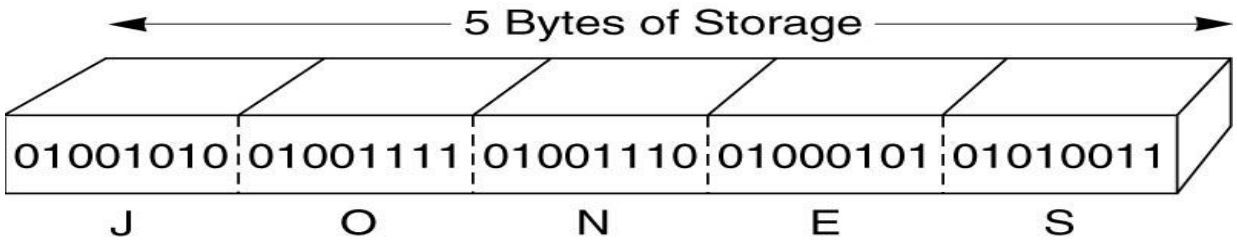
- Floats and doubles are any signed or unsigned numbers having a decimal point as in +10.6255 5. -6.2 0.0 +2.
- Doubles usually use twice the amount of internal storage as the floats.
- May be represented in exponential notation as (+/-)n.dd…d**e**(-)e…e

  1625.00 becomes 1.625**e**3 and .00731 becomes 7.31**e**-3

Data Types and Sizes (cont.)

1. **Integer**
2. ***Floating point***
3. ***Double precision***
4. ***Character***

- Character type represents letters of the alphabets in lower and upper case, ten digits 0-9 and other special symbols.
- Sample of valid character constants are: ‘A’ ‘$’ ‘b’ ‘@’ ‘+’
- Character constants are typically stored in the computer using the ASCII code where every character or symbol is represented by a sequence of 0’s and 1’s that is one byte in length (8 bits)



The ASCII table

ASCII stands for American Standard Code for Information Interchange.

ASCII Table

Decimal	ASCII	Binary
32	blank	00100000
33	!	00100001
34	"	00100010
35	#	00100011
36	$	00100100
37	%	00100101
38	&	00100110
40	(	00101000
41	)	00101001
42	*	00101010
44	,	00101100
45	-	00101101
46	.	00101110
65	A	01000001
66	B	01000010
67	C	01000011
68	D	01000100
69	E	01000101
70	F	01000110
71	G	01000111
72	H	01001000
73	I	01001001
74	J	01001010
75	K	01001011
76	L	01001100
77	M	01001101
78	N	01001110
79	O	01001111
80	P	01010000
81	Q	01010001
82	R	01010010
83	S	01010011
84	T	01010100
85	U	01010101
86	V	01010110
87	W	01010111
88	X	01011000
89	Y	01011001
90	Z	01011010

Decimal	ASCII	Binary
91	[	01011011
92	/	01011100
93	]	01011101
94	^	01011110
95	_	01011111
96	`	01100000
97	a	01100001
98	b	01100010
99	c	01100011
100	d	01100100
101	e	01100101
102	f	01100110
103	g	01100111
104	h	01101000
105	i	01101001
106	j	01101010
107	k	01101011
108	l	01101100
109	m	01101101
110	n	01101110
111	o	01101111
112	p	01110000
113	q	01110001
114	r	01110010
115	s	01110011
116	t	01110100
117	u	01110101
118	v	01110110
119	w	01110111
120	x	01111000
121	y	01111001
122	z	01111010
123	{	01111011
124	|	01111100
125	}	01111101
126	~	01111110

Constants (numeric)

- int can be assigned an integer constant such as 123.
- An example of a float is 123.4
- long represents large values and must be suffixed with an l or L such as 123456789L
- A long float is 123.4L.
  - 123.4 and 123.4L have two different internal representations.

Constants (numeric)

- Constant numbers can be defined in the following way:
  - **Hexadecimal constant:** 0x hexadecimal digits... Where a hexadecimal digit is any digit or any letter A through F or a through f. 0xD is equivalent to 13
  - **Decimal constant:** Any number where the first number is not zero. 13 is the decimal 13
  - **Octal constant:** Any number where the first number must be zero. 013 is octal constant with a decimal value of 11.
  - **Floating constant:** A fractional number, optionally followed by either e or E then the exponent.

Constants (numeric)

- The number may be suffixed by:
  - U or u: Causes the number to be an unsigned integer.
  - L or l: If the number is a floating-point number, then it is a long double, otherwise it is an unsigned long integer.
  - F or f: Causes the number to be a float.

Constants (enum)

- Enumeration allows us to create a series of constant integers.
- The format to create an enumeration is:

  enum identifier {enumerator-list};

  - The Identifier is for identification and may be omitted.
  - The enumerator-list is the list of variables to be created. They are constant integers.
  - Each variable is given the value of the previous variable plus 1.
  - The first variable is given the value of 0 by default but may be initialized.
- Examples:

```
enum {sunday, monday, tuesday, wednesday,thursday, friday, saturday};
enum instructor {me, john=39, mary}; /* me = 0, john = 39; mary = 40*/
```

Constants (characters)

- The char type
  - “char” is a numerical type that is used to represent characters.
  - C provides the character literals that return the numerical value of the character as specified by the ASCII code.
  - Character literals are enclosed in apostrophes, for example:

    char c = ‘D’;

    will assign to c the numerical value of the character D which is 68.
  - Since char is an integer type then it is perfectly legal to use:

    int i = ‘D'; /* This will initialize i to 68 */
  - C also defines a special set of character literals such as newline ‘\n’ and the tab character ‘\t’ which looks like two characters but it is really one.

Declaring Variables

- A variable must be declared (named and assigned a data type) before it is used.
- This is done by using ***declaration statements***. These statements must come before the variable is used and are of the form

  datatype variableName;

  As in:

```
int myAge;
float myIncome;
double myFutureIncome;
```

Declaring Variables

- Setting the value of a variable is called initializing the variable. The statement doing the initialization is called an ***initialization statement***.
  - For example:
    - myAge = 29;
- We could combine a declaration statement with the appropriate initialization statement to get what we call a ***definition statement***.
  - For example:
    - int myAge = 29;
    - char myFirstInitial = 'R';

Meaningful Variable Naming

- You can include type of the variable as part of the name
- Example
  - int iMyAge;
  - char cMyFirstInitial = ‘R’;
  - float fBankBalance;

Initializing to zero or NULL

- Memory locations may contain random data
- Always initialize your variables
  - int = 0;
  - Character set ‘\0’ is known as Null
    - char secondInitial = ‘\0’;
    - Null data is undefined

Constants

- C has a qualifier that tells the application that the value of the variable cannot be changed.
- This qualifier is ***const*** and is used as in:

```
const int myAge = 29;
/* The value of myAge will stay constant at 29 all the time */
```

Arithmetic Operators

Basic Operators:

- The basic operators for performing arithmetic are the same in many computer languages:

+	addition	binary operation	
-	subtraction	binary operation	
*	multiplication	binary operation	
/	division	binary operation	
%	modulus	binary operation	(to capture the remainder)
-	negation	unary operation	

- Binary operations follow the following type return rules:
  1. If the two operands are integers then the result is an integer.
  2. If any operand is a float or a double then the result is a double.

Arithmetic Operators

Integer Division:

- When applied to integers, the division operator / truncates (discards) any remainder.
  - So 5/2 returns 2 and 7/4 returns 1.
- But when either operand is a float or a double, the division operator yields a double precision result.
  - So 5/2.0 returns 2.5, and 7.0/4.0 returns1.75.
- The modulus operator % allows us to capture the remainder when dividing two integers.
  - 5 % 2 is 1; 7 % 4 is 3.
  - (The modulus operator can only be applied to integers.)

Arithmetic Operators

Exponention:

- C does not have an exponentiation operator (typically ^ or **).
  - C has a power function pow(), which we will cover later.
    - for now to square or cube a number, just multiply it by itself as many times as required

Increment and Decrement:

- Incrementing and decrementing are special forms of counting in which we count by 1.
  - Increment uses the special operator ++ and decrement uses the operator --.
  - Example: myAge++;

Assignment operators

The fact that assignment in C is a binary operation has very important implications.

- v1 = v1 + e1; is a valid statement.
- It means add the value of e1 to the current value v1 and store the new result in v1's location

```
sum = 11;
sum = sum + 13;
```

Means the new value stored in sum is 24.

Operator	Meaning
v += i	v = v + i
v -= i	v = v – i
v *= i	v = v * i
v /= i	v = v / i
v %= i	v = v % i
++i	i = i + 1
--i	i = i - 1

Expressions

- An ***expression*** is a series of operands, operators, and function calls that evaluates to a single value.
- For now we will concentrate on expressions containing operators and operands. Rules for forming and evaluating such expressions include:
  1. Binary operators may not follow each other, for example 5+%2 is an invalid expression.
  2. Parenthesis may be used to group operands to remove ambiguity for example 3 + 2*4 may be interpreted as (3+2)*4 = 20 or 3+(2*4)=11.
  3. In C parenthesis may not be used to represent multiplication. Do not use (3+5)(2+6), you should use (3+5)*(2+6).
  4. When arithmetic expressions are not grouped ***precedence rules*** (priority rules) are applied to remove ambiguity.

Expressions

- ***Precedence rules*** (priority rules) are applied to remove ambiguity as follows:
  1. Reduce the expression one step at a time by scanning the expression from left to right.
  2. Multiplication, division, and modulus all have higher precedence than addition and subtraction.
  3. Operations of higher precedence must be performed first.
  4. Operations of same precedence are performed left to right except for unary – which must be performed right to left.
  5. Nested parenthesis are evaluated from the inner most outward.

```
8 + 5 * 7 % 2 * 4 =
8 + 35 % 2 * 4 =
8 + 1 * 4 =
8 + 4 =
12
```

```
3.0 * 4 / 6 + 6 =
12.0 / 6 + 6 =
2.0 + 6 =
8.0
```

Operator Precedence Table
(1, 3, 4 covered so far)

	Description	Represented By
1	Parenthesis	() []
1	Structure Access	. ->
2	Unary	! ~ ++ -- - * &
3	Mutiply, Divide, Modulus	* / %
4	Add, Subtract	+ -
5	Shift Right, Left	>> <<
6	Greater, Less Than, etc	> < =
7	Equal, Not Equal	== !=
8	Bitwise AND	&
9	Bitwise Exclusive OR	^
10	Bitwise OR	|
11	Logical AND	&&
12	Logical OR	||
13	Conditional Expression	?:
14	Assignment	= += -= etc
15	Comma	,

printf() and scanf()

- printf – prints to the screen.
  - Can also accept variables and print their values.
- scanf – gets values from the standard input and assigns them to variables.
- Both rely on #include <stdio.h>

printf Can Print Variable Values

```
printf("I am %d\n", myAge);
```

- The sequence **%d** is a special sequence and is not printed!
- It indicates to `printf` to print the value of an **integer** variable written after the printed string.

scanf gets Input from the user

```
scanf("%lf", &cm);
```

- This statement waits for the user to type in a double value, and stores it in the variable named '`cm`'.
- To get 2 doubles from the user, use –

```
scanf("%lf%lf", &var1, &var2);
```

prinft/scanf Conversion Codes

- A `%<conversion code>` in the printf/scanf string is replaced by the respective variable.
  - `%c` – a character
  - `%d` – an integer, `%u` – an unsigned integer.
  - `%f` – a float
  - `%lf` – a double
  - `%%` - the '`%`' character (in printf)

Example of printf(), scanf()

```
/* Get a length in cm and convert to inches */
#include <stdio.h>

int main()
{
   double cm, inches;

   printf("Please enter length in centimeters: ");
   scanf("%lf",&cm);

   inches = cm / 2.5;
   printf("This is equal to %f inches\n", inches);

   return 0;
}
```

Example 2- Profit Wiz

Enter total revenue: 4500

Enter total cost: 750

Your profit is $3750.00

```
#include <stdio.h>
main()
{
    float revenue, cost;
    revenue = 0; cost = 0;
    printf("\nEnter total revenue: ");
    scanf("%f", &revenue);
    printf("\nEnter total cost: ");
    scanf("%f", &cost);
    printf("\nYour profit is $%.2f\n", revenue - cost);
}
```

The End!