Programming Fundamentals

Module B - Computation

Le The Anh
anhlt161@fe.edu.vn

FPT Education

FPT UNIVERSITY

Objectives

1. Variables
   - Data Types
   - Representation of Integral Values
   - Representation of Floating-Point Data
   - Declarations
2. Basic Memory Operations
   - Constants
   - Assignment Operators
   - Output
   - Input
3. Expressions
   - Arithmetic Expressions
   - Relational Expressions
   - Logical Expressions
   - Shorthand Assignment Operators
   - Mixed Data Types
   - Casting
   - Precedence

Variables

- A variable is a name reference to a memory location, holds data that can change in value during the lifetime of the variable.
- The C language associates a data type with each variable.
- Each data type occupies a compiler-defined number of bytes.

Primitive Data Types

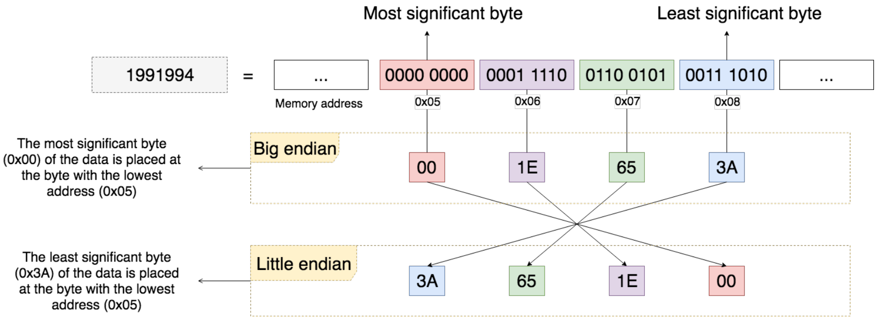
- Typed languages, such as C, subdivide the universe of data values into sets of distinct type.
- A data type defines:
  - how the values are stored and
  - how the operations on those values are performed.
- C has four primitive data types:
  - int (one word): On a 32-bit machine, an int occupies 4 bytes.
  - char (1 byte)
  - float (4 bytes)
  - double (8 bytes)

Qualifiers

- We can qualify the int data type so that it contains a minimum number of bits.
- Qualifiers:
  - short :at least 16 bits
  - long: at least 32 bits
  - long long: at least 64 bits
  - long double: at least 64 bits
- Standard C does not specify that a long double must occupy a minimum number of bits, only that it occupies no less bits than a double.

Representation of Integral Values

- C stores integral values in equivalent binary form.
- Non-Negative values:
  - Intel machines use this little-endian ordering.
  - Motorola machines use big-endian ordering - left to right.



In-Class Practice

Convert the following numbers from current format to the others:

Binary	Octal	Decimal	Hexadecimal
11000101			
	1327		
		1991994	
			1A2FE

In-Class Practice

Convert the following numbers from current format to the others:

Binary	Octal	Decimal	Hexadecimal
11000101	305	197	C5
1011010111	1327	727	2D7
111100110010100111010	7462472	1991994	1E653A
11010001011111110	321376	107262	1A2FE

Negative and Positive Values

- Computers store negative integers using encoding schemes:
  - two's complement notation,
  - one's complement notation, and
  - sign magnitude notation.
- All of these schemes represent non-negative integers identically.
- The most popular scheme is two's complement.

Negative and Positive Values

Two's Complement Notation

- Two's complement notation uses n bits to represent 2^n numbers, ranging from -2^{n-1} to $2^{n-1}-1$. Two bytes can be used to represent 65536 numbers, ranging from -32768 to 32767.
- The most left bit is used to indicate the sign (0: positive numbers, 1: negative numbers)

1000	-8		
1001	-7	0111	7
1010	-6	0110	6
1011	-5	0101	5
1100	-4	0100	4
1101	-3	0011	3
1110	-2	0010	2
1111	-1	0001	1
		0000	0

- -3 + 3 = 1101 + 0011 = 10000 = 0000 (keep only 4 bits) = 0
- -2 + 6 = 1110 + 0110 = 10100 = 0100 = 4

Two's complement notation

Negative decimal to two's complement

- Convert negative decimal number to two's complement notation:
  - represent the absolute value to binary,
  - flip the bits,
  - add one.

Bit #	7	6	5	4	3	2	1	0
92 =>	0	1	0	1	1	1	0	0
Flip Bits	1	0	1	0	0	0	1	1
Add 1	0	0	0	0	0	0	0	1
-92 =>	1	0	1	0	0	1	0	0

- How does it work?
  - a + (-a) = a + (flip(a) + 1) = (a + flip(a)) + 1 = 11...1 + 1 = 100...0 = 0 (remove the most left bit)

Two's complement notation

Two's complement notation to decimal

- If the left most bit is 0 then it's a positive number.
  - Normally convert it to decimal
- If the left most bit is 1 then it's a negative number.
  - flip the bits
  - add one
  - convert to decimal
  - add the sign
- Examples:
  - $01101101 \rightarrow$ decimal?
    - $01101101 \rightarrow 2^6 + 2^5 + 2^3 + 2^2 + 2^0 = 109$
  - $11101101 \rightarrow$ decimal?
    - flip the bits: $\rightarrow 00010010$
    - add one: $\rightarrow 00010011$
    - convert to decimal: $\rightarrow 2^4 + 2^1 + 2^0 = 19$
    - add the sign: $\rightarrow -19$

- Suppose one byte is used to represent the two's complement binary number, what is the two's complement notation of
    - -63
    - -219
- Convert the following binary notation to decimal:
    - 11110101
    - 10111011
    - 01000101

- We can use all of the bits available to store the value of a variable.
  - unsigned short
  - unsigned int
  - unsigned long
  - unsigned long long
- With unsigned variables, there is no need for a negative-value encoding scheme.

Cultural Symbols

- We store cultural symbols using an integral data type.
- We store a symbol by storing the integer associated with the symbol.
- Over 60 encoding sequences have already been defined. We use the ASCII encoding sequence throughout this course.

Encoding Sequence	Full Name	# Bits	Defined In
UCS-4	Universal Multiple-Octet Coded Character Set	32	1993
BMP	Basic Multilingual Plane	16	1993
Unicode	Unicode	16	1991
ASCII	American Standard Code for Information Interchange	7	1963
EBCDIC	Extended Binary Coded Decimal Interchange Code	8	1963

- We use the ASCII encoding sequence throughout this course.

ASCII Encoding Sequence

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20		64	40	@	96	60	`
^A	1	01		SOH	33	21	!	65	41	A	97	61	a
^B	2	02		STX	34	22	"	66	42	B	98	62	b
^C	3	03		ETX	35	23	#	67	43	C	99	63	c
^D	4	04		EOT	36	24	$	68	44	D	100	64	d
^E	5	05		ENQ	37	25	%	69	45	E	101	65	e
^F	6	06		ACK	38	26	&	70	46	F	102	66	f
^G	7	07		BEL	39	27	'	71	47	G	103	67	g
^H	8	08		BS	40	28	(	72	48	H	104	68	h
^I	9	09		HT	41	29	)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B		ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C		FS	60	3C	<	92	5C	\	124	7C	¦
^]	29	1D		GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^-	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂*

In-Class Practice

- What is the ASCII encoding for: '0', 'a', 'A'
- What is the EBCDIC encoding for: '0', 'a', 'A'
- Convert the following binary notation to an ASCII character: 01101101, 01001101
- Convert the following decimal notation to an EBCDIC character: 199, 135

!!! EBCDIC - ASCII

Representation of Floating-Point Data

- Computers store floating-point data using two separate components: an exponent and a mantissa.
- Under IEEE 754, the model for a float occupies 32 bits:

one sign bit	8-bit exponent	23-bit mantissa

  We calculate the value using the formula:

  $$x = \mathtt{sign} * 2^{\mathtt{exponent}} * \{1 + f_1 2^{-1} + f_2 2^{-2} + \cdots + f_{23} 2^{-23}\},$$

  where f_i is the value of bit i and $-126 < \mathtt{exponent} < 127$.
- Under IEEE 754, the model for a double occupies 64 bits:

one sign bit	11-bit exponent	52-bit mantissa

  We calculate the value using the formula:

  $$x = \mathtt{sign} * 2^{\mathtt{exponent}} * \{1 + f_1 2^{-1} + f_2 2^{-2} + \cdots + f_{52} 2^{-52}\},$$

  where f_i is the value of bit i and $-1022 < \mathtt{exponent} < 1023$.
- Refer these tutorials for more details.

Representation of Floating-Point Data

- The limits on float and double under the IEEE standard are:

Type	Size	Significant	Min Exponent	Max Exponent
float	4 bytes	6	-37	38
double	8 bytes	15	-307	308

- The exponent range values in this table are decimal (base 10).
- Note that both the number of significant digits and the range of the exponent are limited.

Declarations
Syntax

- Syntax: data_type identifier [= initial value];
- For example:

```
char section;
int numberOfClasses;
double cashFare = 2.25;
```

- In allocating memory for variables of identical data type, we may group the identifiers in a single declaration and separate them with commas. For example:

```
char section, letter, initial, answer;
int numberOfClasses, children, books, rooms;
double cashFare, money, height, weight;
```

- must start with a letter or underscore (_)
- may contain any combination of letters, digits and underscore (_)
- must not be a C reserved word (key words)
- some compilers allow more than 31 characters, while others do not. To be safe, we avoid using more than 31 characters.

In-Class Practice

- Which of the following is an invalid identifier: whale, giraffe's, camel_back, 4me2, _how_do_you_do, senecac.on.ca, digt3, register
- Select a descriptive identifier for and write a complete declaration for:
    - A shelf of books
    - A cash register
    - A part_time student
    - A group of programs

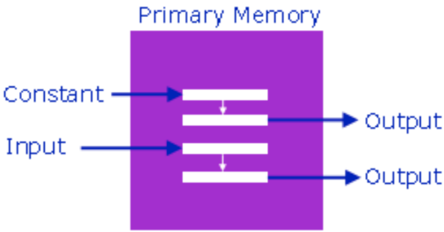
Summary

- Variables
  - Data Types
  - Integral Types
  - Floating-Point Types
  - Declarations

Q&A

Basic Memory Operations

- The C compiler allocates space for program variables in primary memory.
- We work with variables in four principal ways. For example, we can
  - assign a constant value to a variable,
  - assign the value of another variable to a variable,
  - output the value of a variable,
  - input a fresh value into a variable's memory location.



Constants

Numeric

- The compiler embeds constants directly into the program instructions and does not allocate memory for constants.
- Numeric: We identify the data type of a numeric constant by a suffix, if any, on the value itself.

Type	Size	Suffix	Example
int	1 word	none	1456234
long	32 bits	L or l (ell)	75456234L
long long	64 bits	LL or ll (ell ell)	75456234678LL
unsigned		u or U	75456234U
double	2 words	none	1.234
float	32 bits	F or f	1.234F

Constants
Character

- We define a character constant in any one of four ways:

```
// the character itself enclosed in single quotes
char c1 = 'A';
// the corresponding decimal value in the encoding sequence
char c2 = 65;
// the corresponding hexadecimal value in the encoding sequence
char c3 = 0x41;
// the corresponding octal value in the encoding sequence
char c4 = 0101;
```

- We represent special actions or characters that are difficult to write directly by escape sequences:

Character	Sequence	ASCII	EBCDIC
alarm	`\a`	7	47
backspace	`\b`	8	22
form feed	`\f`	12	12
newline	`\n`	10	37
carriage return	`\r`	13	13
horizontal tab	`\t`	9	5
vertical tab	`\v`	11	11
backslash	`\\`	92	*
single quote	`\'`	39	125
double quote	`\"`	34	127
question mark	`\?`	63	111

- We represent a literal string by enclosing it in double quotes. For example, “FPT University\n”

Assignment Operators

- We use the assignment operator to store a value in a program variable. The form of assignment is

  $$\text{variable} = \text{variable, constant or expression}$$

- For example:

```
int age;
age = 18;
```

Output

- To output data to the standard output device, we use a statement of the form

  printf(format string , variable , ... , variable);

- The format string itself consists of the characters to be displayed interspersed with conversion specifiers.

Specifier	Output As A	Use With Data Type
`%c`	character	`char`
`%d`	decimal	`char, int`
`%u`	decimal	`unsigned int`
`%o`	octal	`unsigned char, int, short, long`
`%x`	hexadecimal	`unsigned char, int, short, long`
`%hd`	short decimal	`short`
`%ld`	long decimal	`long`
`%lld`	very long decimal	`long long`
`%f`	floating-point	`float`
`%lf`	floating-point	`double`
`%le`	exponential	`double`

Output

Example

```c
#include <stdio.h>

int main()
{
    int age = 18;
    double cashFare = 2.25;

    printf("His age is %d\nThe cash fare is $%lf\n", age, cashFare);

    return 0;
}
```

Output:

```
His age is 18
The cash fare is $2.250000
```

Input

- The statement to get data from the standard input device:

  scanf(format string , address , ... , address);

- The format string is a literal string. After the format string, we list the memory address of each variable into which we want the input stored. The prefix & denotes 'address of'.
- For example:

```
scanf("%d", &age);
```

Input/Ouput Example

```c
#include <stdio.h>

void main()
{
   int age;
   double cashFare;

   // Input
   printf("Enter the boy\'s age : "); // prompt for age
   scanf("%d", &age);                 // accept age
   printf("Enter the cash fare :");   // prompt for cash fare
   scanf("%lf", &cashFare);           // accept cash fare

   // Output
   printf("His age is %d\nThe cash fare is $%lf\n", age,cashFare);
}
```

In-Class Practice

- Write a program that prompts the user for the amount of money in their pockets, accepts the user input, and displays the amount in the format shown below. If the user enters 4.52, your program displays:

```
How much money do you have in your pockets?  4.52
The amount of money in your pockets is $4.52
```

Summary

- Basic Memory Operations
  - Constants
  - Assignment Operator
  - Output
  - Input

Q&A

Expressions

- A simple expression consists of an operator and operand(s).
- A compound expression consists of several operators and several operands.
- The operands may be variables, constants and/or other expressions.
- Any expression evaluates to a single value, to which we may refer on the right side of an assignment.
- For example: result = (a + b) * c
    - +, *: operators
    - a, b, c: operands

Expressions

- The compiler translates all expressions into sets of simple instructions that the Arithmetic Logic Unit (ALU) can process.
- The ALU can only evaluate the simplest expression that consist of an operator and one or two operands. If there are two operands, they must be of identical data type.
- The ALU receives the operator from the Control Unit. The operator may be
  - arithmetic,
  - relational or
  - logical.

Arithmetic Expressions

- Integral Data Types: The int and char data types accept 5 binary and 2 unary arithmetic operators
  - Binary: +, -, *, /, %
  - Unary: +, -

Arithmetic Expressions

```c
#include <stdio.h>

void main()
{
    int intRight, intLeft, intResult;
    double fptRight, fptLeft, fptResult;

    /* Input */
    printf("Enter an integer : ");
    scanf("%d", &intLeft);
    printf("Enter an integer : ");
    scanf("%d", &intRight);
    printf("Enter a floating-point number : ");
    scanf("%lf", &fptLeft);
    printf("Enter a floating-point number : ");
    scanf("%lf", &fptRight);

    /* Evaluations */
    intResult = intLeft * intRight;
    fptResult = fptLeft * fptRight;

    /* Output */
    printf("%d * %d = %d\n", intLeft, intRight, intResult);
    printf("%le * %le = %le\n", fptLeft, fptRight, fptResult);
}
```

Statistics

- Division typically uses more resources. To avoid division, we multiply rather than divide.
- Moreover, we prefer integral operations to floating-point ones.

Relational Expressions

- Relational expressions compare values and represent conditions with a true or false result.
- The C language interprets the value zero as false and any other value as true.
- Each primitive data type admits 6 relational operators for comparing values of that data type to other values of the same data type.
- Operators: $==, >, <, >=, <=, !=$

Relational Expressions

```c
/* Relational Expressions
 *  relational.c
 *  BTP100
 *  Jan 21 2005
 */

void main( ) {
    int age, childTicket, seniorTicket;

    printf("What is your age ? ");
    scanf("%d", &age);

    childTicket = age <= 12;
    seniorTicket = age >= 65;

    printf("You need a child  Ticket (1 for yes, 0 for no) : %d\n", childTicket);
    printf("You need a senior Ticket (1 for yes, 0 for no) : %d\n", seniorTicket);
 }
```

Logical Expressions

- Logical expressions compare conditions and yield true or false results.
- We use logical operators to express compound conditions.
- The C language has 2 binary logical operators (&&, ||) and 1 unary operator (!).

Logical Expressions

```c
void main( )
{
   int age, atSchool, childTicket, studentTicket, adultTicket, seniorTicket;

   printf("What is your age ? ");
   scanf("%d", &age);
   printf("Are you at school (1 for yes, 0 for no) ? ");
   scanf("%d", &atSchool);

   childTicket = age <= 12;
   studentTicket = age > 12 && age <= 19 && atSchool == 1;
   seniorTicket = age >= 65;
   adultTicket = !childTicket && !studentTicket && !seniorTicket;

   printf("You need a child   Ticket (1 for yes, 0 for no) : %d\n", childTicket);
   printf("You need a student Ticket (1 for yes, 0 for no) : %d\n", studentTicket);
   printf("You need a senior  Ticket (1 for yes, 0 for no) : %d\n", seniorTicket);
   printf("You need an adult  Ticket (1 for yes, 0 for no) : %d\n", adultTicket);
}
```

Shorthand Assignment Operators

- The C language includes a set of shorthand assignment operators, which combine arithmetic expressions with assignments.

Operator	Shorthand	Longhand	Meaning
+=	age += 4	age = age + 4	add 4 to age
-=	age -= 4	age = age - 4	subtract 4 from age
*=	age *= 4	age = age * 4	multiply age by 4
/=	age /= 4	age = age / 4	divide age by 4
%=	age %= 4	age = age % 4	remainder after age/4
++	age++ or ++age	age = age + 1	increment age by 1
--	age-- or --age	age = age - 1	decrement age by 1

Shorthand Assignment Operators

- The pre-fix operator changes the value of the operand before the value is used, while the post-fix operator changes the value of the operand after the value has been used.

```c
int a = 1;
int b = a++;
int c = ++a;

printf("%d %d %d", a, b, c); // 3 1 3
```

Mixed Data Types

- Although the ALU does not perform operations on operands of differing data type directly, C compilers can interpret expressions that contain operands of differing data type.
- If a binary expression contains operands of differing type, a C compiler changes the data type of one of the operands to match the other.
- Data type hierarchy:

double	higher
float	...
long	...
int	...
char	lower

Mixed Data Types

Assignment Expressions

- If the data type of the variable on the left side of an assignment operator differs from the data type of the right side operand, the compiler
  - promotes the right operand to the data type of the left operand if the left operand is of a higher data type than the right operand,
  - truncates the right operand to the data type of the left operand if the left operand is of a lower data type than the right operand.
- For example:

```
float a = 1.6;
int b = a;
float c = b;

printf("%.1f %d %.1f", a, b, c); // 1.6 1 1.0
```

Mixed Data Types

Arithmetic and Relational Expressions

- If the operands in an arithmetic or relational expression differ in data type, the compiler promotes the value of lower data type to a value of higher data type before implementing the operation.

	(Right Operand)				
(Left Operand)	double	float	long	int	char
double	double	double	double	double	double
float	double	float	float	float	float
long	double	float	long	long	long
int	double	float	long	int	int
char	double	float	long	int	char

Table 1: Data Type of Promoted Operand

- For example:
  - 1034 * 10L yields 10340L (a long result)
  - 1034 * 10.f yields 10340.0f (a float result)

Casting

- Casting is the way to change the data type of any constant or variable.
- Syntax: (desired data type) constant or variable.
- For example:

```c
int a = 4, b = 3;
double c, d;

c = a / b;
d = (double) a / b;

printf("%d %d %.2lf %.2lf", a, b, c, d); // 4 3 1.00 1.33
```

Precedence

- The rules of precedence define the order in which a compiler must decompose a compound expression.
- The compiler evaluates the first operation with the operator that has highest precedence.

Operator	Evaluate from
++ -- (post)	left to right
++ -- (pre) + - & ! (all unary)	right to left
(data type)	right to left
* / %	left to right
+ -	left to right
$<$ $\leq$ $>$ $\geq$	left to right
== !=	left to right
&&	left to right
||	left to right
= += -= *= /= %=	right to left

Precedence

- Use ( ) to instruct the compiler to evaluate the expression within the parentheses first. For example:
  - $2 + 3 * 5 \rightarrow 2 + 15 = 17$
  - $(2 + 3) * 5 \rightarrow 5 * 5 = 25$

Summary

- Expressions
  - Arithmetic
  - In-Class Problem
  - Statistics
  - Relational
  - Logical
  - Shorthand Assignment Operators
  - Mixing Data Types
  - Casting
  - Precedence

Q&A