EEE101: C Programming & Software Engineering I

Lecture 3: Variables and their Types

Dr. Rui Lin/Dr. Mark Leach
Office: EE512/EE510
Email: rui.lin/mark.leach@xjtlu.edu.cn
Dept. of EEE XJTLU

Outline of Today's Lecture (week 3)

- Numbers in the computer
- Variables and type conversions
- Characters, Arrays and Strings
- Basic Input and Output
- Operators (that's mathematical functions)
- But first……..

Numbers in the Computer

ON - 1

OFF - 0

- Remember…in a computer all numbers are binary.
- Convenient for computer **because?**…
- not for us – we like decimal **because?**
- Hexadecimal (base 16) is more convenient….

Numbers in the Computer

Dec	Binary	Hex	Dec	Binary	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Easier to read than binary, one byte = 8 bits or 2 hex digits

Decimal 75 = 01001011 = 4B

You should be able to convert between the number bases.

Variables - Types and Conversions

Variables

- Remember – a variable is a *named memory location* that can hold a value (in binary)
- In C (and most languages) a variable must be declared before use.
- Declaration does 2 things:
  - Determines the type of variable
  - Reserves specific memory space for that variable
- Using the keyword **const** before the variable declaration prevents its value from being changed.

e.g. **const float** pi=3.141; or

const char msg[]="warning";

Enumeration Constant (enum)

- Another kind of constant is used to assign a sequence of numbers to names
- Called an enumeration constant with the keyword **enum** e.g.

enum boolean {No, Yes}; assigns No=0 and Yes=1

enum month {Jan=1, Feb, Mar,…Nov, Dec};

assigns Jan=1, Feb=2… (Note you have to write all months)

- Advantage over **#define** – offers type check

Data Types and Ranges

Data Type	Description	Bytes	Range
short	short integer	2	-32,768 to 32,767
unsigned short	positive short integer	2	0 to 65,535
int	integer	2 or 4	see short or long
unsigned int	positive integer	2 or 4	see unsigned short or long
long	long integer	4	-2,147,483,648 to 2,147,483,647
unsigned long	positive long integer	4	0 to 4,294,967,295
float	single precision real number	4	3.4×10^{-38} to 3.4×10^{38} (6 digits of precision)
double	double precision real number	8	1.7×10^{-308} to 1.7×10^{308} (10 digits of precision)
long double	double precision real number	12	3.4×10^{-4931} to 3.4×10^{4931} (10 digits of precision)
char	character	1	-128 to 127
unsigned char	positive character	1	0 to 255

Computational Problems

- Real and integer numbers are stored differently
- How does the computer deal with this e.g.

```
float answer;
int x=24, y=10;
answer=x/y;
```

What value is stored in answer?

Computational Problems

- Real and integer numbers are stored differently
- How does the computer deal with this e.g.

```
float answer;
int x=24, y=10;
answer=x/y;
```

What value is stored in answer? 2.0

How can the correct value be obtained?

Type Conversions

- This refers to the changing of one data type to another
- Two type conversions *implicit* and *explicit*
- **Why not just store everything in the same format?**

Type Conversions

- This refers to the changing of one data type to another
- Two type conversions *implicit* and *explicit*
- **Why not just store everything in the same format?**

  Data can be stored in most compact form and converted when needed
- Disadvantage

  Type mismatches can be missed by the compiler

Implicit Conversions

- When types are mixed then type conversions that can be, are performed automatically.
- Remember that a **char** is just a small integer value and can be used in mathematical operations.
- There are some rules for these conversions:

Implicit Conversions

- **int** k=5, m=4, n; **float** x=1.5, y=2.1, z;

Context	Example	Explanation
Expression with binary operator and operands of different numeric types	k+x; value 6.5	Value of **int** variable k is converted to type **float** before operation
Assignment statement with **float** target and **int** expression	z=k/m; expression value is 1 value of z is 1.0	Expression is evaluated first. Result is converted to double for assignment
Assignment statement with **int** target and **float** expression	n=x*y; expression value is 3.15 value of n is 3	Expression is evaluated first. Result is converted to **int** for assignment. Fraction is lost.

Explicit Conversions

- You can force the use of a particular type and the implicit conversion can be ignored
- Use a cast operator to convert the type
- Note the cast does not change a value stored in a variable, it just changes the type for the calculation.

```
int age;
age=1.2+5.978; automatic result is 7
age=(int)1.2+(int)5.978; casts convert values first
age=?
```

Explicit Conversions

- You can force the use of a particular type and the implicit conversion can be ignored
- Use a cast operator to convert the type
- Note the cast does not change a value stored in variable, it just changes the type for the calculation.

```
int age;
age=1.2+5.978; automatic result is 7
age=(int)1.2+(int)5.978; casts convert values first
age=6
```

Characters, Arrays and Strings

Characters and Arrays

- Characters

  Consist of any printable or non-printable character in the computers character set include: lowercase/uppercase letters, decimal digits, special characters and escape sequences.

  Generally stored in a single byte (8-bits)

- Arrays

  Ordered sequences of same type data elements

  Simply put, several memory cells in a row given the same name.

Defining an Array

- How to declare an array:

```
char name[20];
```

- This declares an array called name with 20 elements (i.e. name can store 20 characters)
- Brackets [] indicate an array
- An array MUST have a dimension (number of elements)
- Examples:

```
int attendance[15];
float  daily_temperature[30];
```

What do these mean??

Accessing Array Elements

- How can I use each element of my array
- The first element is always **0**
- **DO NOT** exceed array bounds (no one will check!)
- Consider:

```
float price[20]; /*declare an array 20 elements*/
price[0]=12.12; /*assign value to first element*/
price[1]=13.13; /*assign value to second element*/
…
price[20]=12.34; /*assign value to element 20?*/
```

What will happen here?

String Fundamentals

- A string is an array of characters ending with the NULL character or '\0'
- Can be written with double quotes "I am a string"
- Can be assigned when array is declared:
  - **char** firstname[] = "Walter";
  - **char** lastname[] = "White";
  - **char** fullname[] = "Walter White";
- **#define** can be used to create a string
  - **#define** TV "Breaking Bad"

String Fundamentals

- A string can also be defined by specifying individual characters:
  - **char** colour[] = {‘g’,’r’,’e’,’e’,’n’,’\0’};
  - This shows that each character is stored in its own element.

what would this print?

printf(“the second character is %c”,colour[2]);

String Library

- There is a library of string functions *string.h*
- Some examples:
  - strlen() – finds the length of a string
  - strcmp() – compares two strings character by character
  - strcpy() – copies a string from one array to another

```
#include<string.h>
#include<stdio.h>
#define NAME “Mark Leach”

main(){
printf(“My name has %d characters”,strlen(NAME));
}
```

Basic I/O Functions And Operations

Basic Input and Output

- Standard input functions (in *stdio.h*)
  - getchar() reads a character
  - scanf() input type must be specified
  - gets() reads strings
- Standard output functions (in *stdio.h*)
  - putchar() outputs a character
  - printf() output type must be specified
  - puts() outputs a string

The scanf() function (1/2)

- reads data from the standard input device *stdin* (usually the keyboard) and stores it in a variable

- General syntax:

  scanf("format specifier", &variable);

- The ampersand (&)
  - Specifies the memory address of the variable
- example:

```
int age;
printf("enter your age:");
scanf("%d",&age);
```

The scanf() function (2/2)

- Common format specifiers used in printf() and scanf() functions

Type	Specifier
int	%d
float	%f
double	%lf
char	%c
string	%s

- Add more specifiers to enter more than one argument:

```
float height, weight;
scanf("%f%f",&height,&weight);
```

getchar() and putchar()

- Single character reading and writing, examples:

```
#include<stdio.h>
main(){
char my_char;
printf("please type a character: ");
my_char=getchar();
printf("\n you typed the character: ");
putchar(my_char);
}
```

gets() and puts()

- Multiple character reading and writing
- scanf() reads strings using the specifier %s , however it cannot accept the space
- If the string to be read contains a space, the gets() function MUST be used

Example (1/2)

```
#include<stdio.h>
main()
{
char string1[50], string2[50];
printf("Enter a string less than 50 characters with spaces: \n");
gets(string1);
printf("you entered:");
puts(string1);
printf("Enter the same string\n");
scanf("%s",string2);
printf("you entered:");
puts(string2);
}
```

Example (2/2)

Sample output:

Enter a string less than 50 characters with spaces:

hello class

You entered: hello class

Enter the same string:

hello class

You entered: hello

Operators in C (Unary)

- Unary operators (involve one variable)

C operation	Operator	Example
Positive	+	a=+3
Negation	-	b=-a
Increment	++	i++
Decrement	--	i--

- The first assigns positive 3 to a
- The second assigns the negative of a to b
- i++ is equivalent to i = i + 1
- i-- is equivalent to i = i - 1

Pre/Post Increment/Decrement

- ++i and i++ or --i and i—
- These are different, the location of the operation (++ or --) decides when the operation happens
- Operation before variable is a pre operation
- Operation after variable is a post operation

```
int a=9;
printf("%d\n",a++);
printf("%d",a);
```

What would the output would be?

Pre/Post Increment/Decrement

What about in this case?:

```
int a=9;
printf("%d\n",++a);
printf("%d",a);
```

Assignment Operator

- Assignments can be written in a more shorthand way:

  i = i+2 is the same as i += 2

- the shorthand is denoted var op= value

Question: x*= y+1

Is this equal to x=x*y+1 or x=x*(y+1)?

Relational Operations

- Used to Compare Expressions:

Operator	*Meaning*
<	is **less** than
<=	is **less** than **or equal** to
!=	is **not equal**
==	is **equal to**
>	is **greater** than
>=	is **greater** than **or equal** to

Logical Operations

Used to combine one or more relational expressions

```
if ( (number>6) && (number<12) )
   printf("You are close!");
else
   printf("You Loose!");
```

Complex test expression – tests whether number lies within a range

Operator	*Meaning*
&&	*Logical "**and**":* True if **both** arguments are true
||	*Logical "**or**":* True if **one or both** arguments are true
!	Logical "**not**": changes True in False and False in True

Logical Operations

Table 4.3 The && Operator (and)

operand 1	operand2	operand 1 && operand2
nonzero (true)	nonzero (true)	1 (true)
nonzero (true)	0 (false)	0 (false)
0 (false)	nonzero (true)	0 (false)
0 (false)	0 (false)	0 (false)

Table 4.4 The || Operator (or)

operand l	operand2	operand 1 || operand2
non zero (true)	nonzero (true)	1 (true)
nonzero (true)	0 (false)	1 (true)
0 (false)	nonzero (true)	1 (true)
0 (false)	0 (false)	0 (false)

Table 4.5 The ! Operator (not)

operand l	!operand 1
nonzero (true)	0 (false)
0 (false)	1 (true)

Precedence – Relational Operators

Assignment operators	Relational operators	Arithmetic Operators

`x>y+2` means the same as `x>(y+2)`

`x=y>2` means the same as `x=(y>2)`

Precedence – Logical Operators

Arithmetic operators	!	()

Lower — Precedence — Higher

Assignment operators	||	&&	Relational operators

```
x=3 > 5 && 10 > 23 || 4 > 2
```

means the same as

```
x=((3 > 5) &&(10 > 23)) || (4 > 2)

x=((False && False) || True)
```

Bitwise Operators

- The following are bitwise operators:

  &, |, ^, <<, >>, ~

- What do they mean? What is the difference between these operators and the relational operators?

- They are usually used for C programming for embedded systems.

Quiz

- Which of the following evaluates to **true**?
- Assume: P = true, Q = false, R = true

1) (!P || R) && Q
2) !(Q && R) && P
3) !(P && R) || Q
4) P && !Q && !R

Quiz

- Can you work out what the following logical expression is?

  i<lim-1 && (c=getchar()) != '\n' && c != EOF

  What will happen, if we remove the parentheses surrounding c=getchar()?

- The unary negation operator "!" converts a non-zero operand into 0 and a zero into 1.

  Can you tell, assuming valid is 0, what the following statement does?

  ```
  if (!valid)
  ……;
  ```

  Can you rewrite this statement using another relational operator?

I know that was long…but…
Thank you again for your attention

See you in the lab ☺