Introduction to Scientific and Engineering Computation

(BIL 102E)

LECTURE 4

HANDLING DATA

- **integer** numbers vs. **real** numbers
- Arithmetic operations
- Priority of operations
- Operations on mixed data types
- Handling **character** strings

Integers

Integers are whole numbers without any decimal part. For example, 1, 0, 34, -12 are all integers.

There is a limited space allocated for representing numbers.

Integers are contained in memory as groups of 2 or 4 bytes. (that is as 16 bits or 32 bits)

The binary representation	Decimal	Integer
0000 0000 0000 0000	0	0
0000 0000 0000 0001	1	1
0000 0000 0000 0010	2	2
1111 1111 1111 1111	65535	-1
1111 1111 1111 1110	65534	-2

Real (Floating Point) Numbers

These are numbers with a fractional part.

For example, 2.3, -14.356, 54.0 are real numbers.

Exponential form can be used to represent real numbers. For example, 1e4 → 10000, 2.5e-2 → 0.025

These numbers are contained in memory in two parts: mantissa and exponent. For example,

$2.3 \Rightarrow 23 \times 10^{-1} \Rightarrow$ 0000 0000 0001 0111 (mantissa)

1111 1111 (exponent)

The number of bits used for the mantissa part and exponent part can change from computer to computer.

Comparison of integer numbers and real numbers

•An integer cannot have a fraction, whereas a real number can.

•On a 32 bit computer an integer can have values between about –2 x 10^9 (2^{31}) to +2 x 10^9 (2^{31}-1), whereas a real number can be between – 10^{38} to +10^{38} with 7 or 8 significant digits.

•So why use integers?

•Integers take **less space in memory** compared to real numbers.

•Arithmetic operations on integers are much **faster** than those on real numbers.

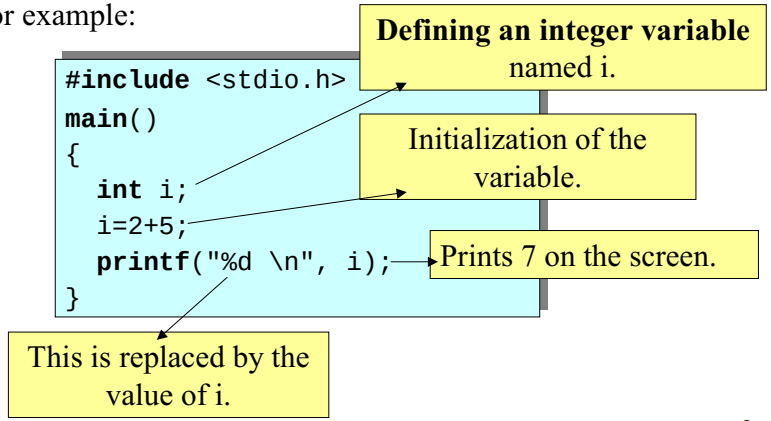
Don't forget that **real** numbers are approximations. For example, the number π can be represented upto 8 significant digits as a **real** number.

Variables

We use variables to store data.

In C programming language, we need to define the names and types of variables before starting using them.

For example:



Declaring **real** (floating point) and **integer** Variables

In C language the syntax for declaring a variable is

type name;

or

type name1,name2, ... ;

So in order to declare an integer variable called *k* we type

int k;

And to declare three real numbered variables with labels *x,y,z*

float x,y,z;

Reserved Words

There are a number of names called **reserved words (keywords),** which may not be used for any purpose other than they are intended to.

There are 32 keywords in ANSI C:

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**

Identifiers

Other names chosen by the programmer are called **identifiers**. The names of variables, functions etc are all identifiers (they identify an entity in memory).

There are several syntax rules an identifier has to obey:

1. It consists of characters containing the letters (A-Z and a-z), the digits (0-9) and the underscore character (_)
2. It cannot start with a digit.
3. It is case sensitive.
4. If the identifier consists more than 31 characters, all characters beyond the 31st character may be ignored by any given compiler.
5. Reserved words cannot be used as identifiers.

An identifier should be chosen to indicate the meaning of its use.

Statements and Statement Blocks

In language C, statement is a complete instruction, ending with a semicolon.

For example,

i = 2+3;

return 0;

printf("Hi");

are all statements.

Although not recommended, it is possible to have more than one statement on a line.

A group of statements can form a *statement block* that starts with an openning brace ({) and ends with a closing brace (}).

A statement block is treated as a single statement by the C compiler.

For example, the main() function is formed by a statement block.

Assignments

The assignment statement in C has the following syntax

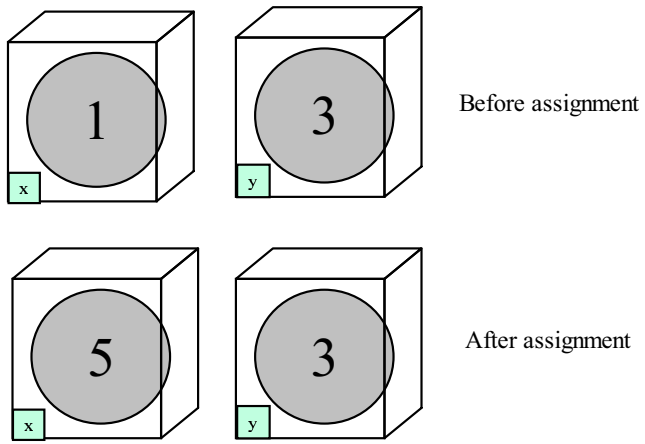
name = expression;

Here *name* is the name of a variable and *expression* is an arithmetic, or other, expression. If necessary expression is first calculated to determine the value to be assigned to the variable. For example,

x = y+2;

Do not confuse the equation sign here with mathematical equations. This assignment could be read in plain English as

"Let x be y plus 2" or similarly "Assign y plus 2 to x".



x = y + 2;

In an assignment statement the same variable can appear in both sides of the statement. For example,

x = x+1;

is a meaningful assignment and would cause the value of x to be incremented by 1. Similarly

x = y * x;

Would cause the value of x to be multiplied by the value of y.

Arithmetic Operations

There are 5 arithmetic operations

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder

Note that division of integers result in an integer.

5/3 → 1

7/8 → 0

(integer truncation)

Also called as modulus

5 % 3 → 2

14 % 7 → 0

Priority of Operations

There can be more than one arithmetic operations in an arithmetic expression.

Operations in an arithmetic expression are done from left to right unless the operation on the right has a priority over the operation over left.

Operator	**Priority**
*, / and %	High
+ and -	Low

Examples

2+3*4

→ 2 + (3 * 4)

→ 2 + 12

→ 14

9/5*10%2+2

→(9/5)*10%2+2

→(1*10)%2+2

→(10%2)+2

→0+2

→2

Priorities can be overwritten by the use of parentheses (). For example,

(2.1+2.1)/0.7 → 6.0

Use parentheses when you are not sure to be on the safe side, and to increase the *readability* of your code. You can also add spaces between variables and operators so as to increase the readability.

When there are variables in the expression first the values of variables are replaced in the expression and then the calculation takes place.

For example

float x,y;
x=2.1;
y=4.2;
x=x+y/x;

assigns 4.1 to x.

Operations on Mixed Data

1. When an operation takes place on a real number and an integer, the integer is first converted to a real number so the result is real.
2. When an integer is assigned to a real number, it is simply converted to a real number.
3. When a real number is assigned to an integer, it is truncated to an integer.

BONUS QUESTION!

What is the output of the following program?

```
#include <stdio.h>
main()
{
  float x;
  int i,j;
  i=2;
  j=4;
  x=1.0e-1;
  i=(j+i) / j * (i+x);
  printf("%d \n",i);
}
```


STORING CHARACTERS

Characters are stored as 8 bits of binary data.

For example,

A → 0100 0001 (decimal 65)
B → 0100 0010 (decimal 66)
C → 0100 0011 (decimal 67)
D → 0100 0100 (decimal 68)
…
a → 0110 0001 (decimal 97)
b → 0110 0010 (decimal 98)
c → 0110 0011 (decimal 99)

The numbers corresponding to the characters are usually called the ASCII codes.

Declaration of **character** Variables

A variable that can represent different variables is called a character variable.

Character variables can be declared using the following syntax:

char *name1, name2, ...* ;

For example,

char ch1,ch2;

Defines two character variables ch1 and ch2.

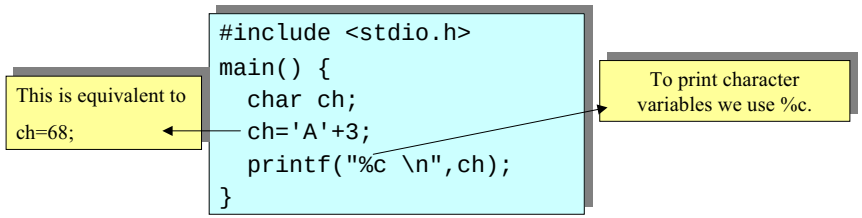
It is then possible, for example

ch1='A';
ch2='a';

Character constants are enclosed in single quotes (')

Giving values to character variables

- It is possible to assign character constants such as 'A', 'B', 'C', 'D' to character variables.
- It is also possible to assign the ASCII codes of the characters to character variables. The C programming language actually treats character variables as numeric variables: For example, the following program prints D on the screen.



Note that by using the escape character (\) it is possible to denote special characters such as the new line character (\n) and the carriage return character (\r).

Giving Initial Values to Variables

It is possible to initialize variables while declaring them. The syntax is as follows:

type var_name = initial_value;

type var1=init1, var2=init2, var3, var4, ...

For example, instead of

```
int i;
i = 5;
```

it is possible to use

```
int i=5;
```

The following statements are allowed in C:

```
int   i=5, j=2, k, m=3;
char  ch1='A', ch2='B', ch3=68, c;
float a, b, f=2.5, g=9.8;
```


Combining Arithmetic Operators with =

The following abbreviations are possible:

`x += y;` is equivalent to `x = x + y;`

`x -= y;` is equivalent to `x = x - y;`

`x *= y;` is equivalent to `x = x * y;`

`x /= y;` is equivalent to `x = x / y;`

`x %= y;` is equivalent to `x = x % y;`

`z *= x + y;` → `z = z*x + y;`

A number can be negated using the **unary minus operator** (-).

For example,

`z = x - -y;` → `z = x – (-y);` → `z = x + y;`

Incrementing by one is abbreviated by unary ++ operator and decrementing by one is abbreviated by unary -- operator.

`x++;` → `++x;` → `x = x + 1;`

`x--;` → `--x;` → `x = x – 1;`

Comments

In a C program anything put in between /* and */ are *comments*. These are not considered by the compiler and has no effect on the execution of the program. These, however, help the future programmers to understand what that particular part of the program is doing. Therefore, they increase the *readability* and *portability* of the programs.

- Effective comments are as important as effective programs.
- Use comments as much and as sensible as possible.

```
#include <stdio.h>
main()
{
  float x,v;  /* x shows the distance and v is the velocity */
  v = 5.0;
  x = v*3.0;   /* Calculate the distance covered for 3 seconds */
  printf("%d \n",x);
}
```


Common Syntax Errors

- **float** r
- **int** printf
- k = 1,234;
- z = x y;
- z = x++y;
- z = 2/(x*(5+2);
- z = 2/5+1*x
- ch = ‘g
- ch = ‘hello’;