CS1100 – Introduction to Programming

Lecture 4

Instructor: Shweta Agrawal (shweta.a@cse.iitm.ac.in)

Assignment operator =

Form: variable-name = expression

- z = x+y
- x+y = z Incorrect form

Assignment operator =

Form: variable-name = expression

- z = x+y
- x+y = z Incorrect form
- Assignment between different data types.
  - What happens if you assign float to int and vice versa?

Assignment operator =

Form: variable-name = expression

- z = x+y
- x+y = z Incorrect form
- Assignment between different data types.
  - What happens if you assign float to int and vice versa?
- Multiple assignments.
  - x = y = z = (a + b);
  - evaluations happen right to left.

Assignment operator =

Form: variable-name = expression

- z = x+y
- x+y = z Incorrect form
- Assignment between different data types.
  - What happens if you assign float to int and vice versa?
- Multiple assignments.
  - x = y = z = (a + b);
  - evaluations happen right to left.
- x = x + 10 can be written as x += 10;
- instead of +, we can also have -, *, /, %

Exercises

Write a program that reads an integer from the input and prints 0 if the integer is even and 1 if the integer is odd.

Write a program that takes as input a 3 digit integer, separates the digits of the integer and prints the individual digits separated by spaces.
For example if the input is 194, then your program must print
1 9 4

Integers in C and Storage

- We have used **int** and **float** data types till now.

Integers in C and Storage

- We have used **int** and **float** data types till now.
- An integer variable is assigned 2 bytes (16 bits) to be stored. (Sometimes 4 bytes).

Integers in C and Storage

- We have used **int** and **float** data types till now.
- An integer variable is assigned 2 bytes (16 bits) to be stored. (Sometimes 4 bytes).
- In the 2s complement form this allows storage of values from

$$-2^{15} \text{ to } 2^{15} - 1$$

Integers in C and Storage

- We have used **int** and **float** data types till now.
- An integer variable is assigned 2 bytes (16 bits) to be stored. (Sometimes 4 bytes).
- In the 2s complement form this allows storage of values from

$$-2^{15} \text{ to } 2^{15} - 1$$

$$-32,768 \text{ to } 32,767$$

- There are limits to representation - we better choose the right type.

Integers in C and Storage

- We have used **int** and **float** data types till now.
- An integer variable is assigned 2 bytes (16 bits) to be stored. (Sometimes 4 bytes).
- In the 2s complement form this allows storage of values from

$$-2^{15} \text{ to } 2^{15} - 1$$

$$-32,768 \text{ to } 32,767$$

- There are limits to representation - we better choose the right type.
- What other data type can we use to store integers?

Integers in C and Storage

- We have used **int** and **float** data types till now.
- An integer variable is assigned 2 bytes (16 bits) to be stored. (Sometimes 4 bytes).
- In the 2s complement form this allows storage of values from

$$-2^{15} \text{ to } 2^{15} - 1$$

$$-32,768 \text{ to } 32,767$$

- There are limits to representation - we better choose the right type.
- What other data type can we use to store integers?
- **unsigned int**, **long**, **unsigned long**.

unsigned int

- Typically 4 bytes storage.
- Output an unsigned int: printf("%u", x);
- Input an unsigned int: scanf("%u", &x);
- Storage: binary format.

The Integers - The detailed Chart

int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

char

- Typically 1 byte storage.
- Every character has a unique code assigned to it (ASCII code).
  A = 65, B = 66

char

- Typically 1 byte storage.
- Every character has a unique code assigned to it (ASCII code).
  A = 65, B = 66
- Output a character: printf("%c", x);
- Input a character: scanf("%c", &x);

float

- Typically 4 bytes storage.
- Output a float: printf("%f ", x);
- Input a float: scanf("%f ", &x);
- How are fractions stored?

Binary vs decimal fractions

- $(10.11)_2 = (1 \times 2) + (0 \times 1) + (1 \times \frac{1}{2}) + (1 \times \frac{1}{2^2}) = (2.75)_{10}$

Binary vs decimal fractions

- $(10.11)_2 = (1 \times 2) + (0 \times 1) + (1 \times \frac{1}{2}) + (1 \times \frac{1}{2^2}) = (2.75)_{10}$
- $(0.90625)_{10} = (\quad)_2$
- $(0.9)_{10} = (\quad)_2$

Decimal Fraction → Binary Fraction (1)

Convert $(0.90625)_{10}$ to binary fraction

	0.90625	
	× 2	
	0.8125	+ integer part
1	× 2	
	0.625	+ integer part
1	× 2	
	0.25	+ integer part
1	× 2	
	0.5	+ integer part
0	× 2	
	0	+ integer part 1

$0.90625 = \frac{1}{2}(1+0.8125)$
$= \frac{1}{2}(1+\frac{1}{2}(1+0.625))$
$= \frac{1}{2}(1+\frac{1}{2}(1+\frac{1}{2}(1+0.25)))$
$= \frac{1}{2}(1+\frac{1}{2}(1+\frac{1}{2}(1+\frac{1}{2}(0+0.5))))$
$= \frac{1}{2}(1+\frac{1}{2}(1+\frac{1}{2}(1+\frac{1}{2}(0+\frac{1}{2}(1+0.0)))))$
$= ½+1/2^2+1/2^3+0/2^4+1/2^5$
$= (0.11101)_2$

Thus, $(0.90625)_{10} = (0.11101)_2$

Decimal Fraction → Binary Fraction (2)

Convert $(0.9)_{10}$ to binary fraction

0.9			
× 2			
0.8	+	integer part	1
× 2			
0.6	+	integer part	1
× 2			
0.2	+	integer part	1
× 2			
0.4	+	integer part	0
× 2			
0.8	+	integer part	0

For some fractions, we do not get 0.0 at any stage! These fractions require an infinite number of bits! Cannot be represented exactly!

Repetition

$(0.9)_{10} = 0.1110011001100 1100 \ldots = 0.\overline{11100}$

Binary vs decimal fractions

- $(10.11)_2 = (1 \times 2^1) + (0 \times 2^0) + (1 \times \frac{1}{2}) + (1 \times \frac{1}{2^2}) = (2.75)_{10}$
- $(0.90625)_{10} = (0.11101)_2$
- $(0.9)_{10} = (0.11100\color{red}{11100}11100..)_2$

Fixed point vs floating point representation

Fixed point

- Position of radix point is fixed and is same for all numbers.
- Lets say we have 3 digits after radix point.

Fixed point vs floating point representation

Fixed point

- Position of radix point is fixed and is same for all numbers.
- Lets say we have 3 digits after radix point.
- $(0.120 \times 0.120)_{10} = (0.014)_{10}$
- A digit is lost.

Floating point

Fixed point vs floating point representation

Fixed point

- Position of radix point is fixed and is same for all numbers.
- Lets say we have 3 digits after radix point.
- $(0.120 \times 0.120)_{10} = (0.014)_{10}$
- A digit is lost.

Floating point

- $1.20 \times (10)^{-1} \times 1.20 \times (10)^{-1} = 1.44 \times (10)^{-2}$
- Wider range of numbers can be represented.
- IEEE standard: 32 bits are split as follows:
  - First bit for sign.
  - Next 8 bits for exponent.
  - Next 23 bits for mantissa (fractional part).

Fixed point vs floating point representation

Fixed point

- Position of radix point is fixed and is same for all numbers.
- Lets say we have 3 digits after radix point.
- $(0.120 \times 0.120)_{10} = (0.014)_{10}$
- A digit is lost.

Floating point

- $1.20 \times (10)^{-1} \times 1.20 \times (10)^{-1} = 1.44 \times (10)^{-2}$
- Wider range of numbers can be represented.
- IEEE standard: 32 bits are split as follows:
  - First bit for sign.
  - Next 8 bits for exponent.
  - Next 23 bits for mantissa (fractional part).
  - $(-39.9)_{10} = (-100111.11100)_2 = (-1.0011111100)_2 \times 2^5$.

Floats - different types

Type	Storage size	Value range
float	4 byte	1.2E-38 to 3.4E+38
double	8 byte	2.3E-308 to 1.7E+308
long double	10 byte	3.4E-4932 to 1.1E+4932

Output floats in C

printf(" %w.p f ", x);

- w.p is optional.
- w : total width of the field.
- p : precision (digits after decimal).

Output floats in C

printf(" %w.p f ", x);

- w.p is optional.
- w : total width of the field.
- p : precision (digits after decimal).

```
#include<stdio.h>
main() {

    float x = 2.00123;
    printf ("x = %5.4f\n", x);
    printf ("x = %8.7f\n", x);
}
```

Circumference of circle

```
#include<stdio.h>

main() {
    float radius;
    float circum;

    printf("Enter radius : ");
    scanf("%f", &radius);
    circum = 2* (22.0/7) * radius;

    printf ("radius = %f, circum = %f\n", radius, circum);
}
```

Circumference of circle

```
#include<stdio.h>

main() {
    float radius;
    float circum;

    printf("Enter radius : ");
    scanf("%f", &radius);
    circum = 2* (22.0/7) * radius;

    printf ("radius = %f, circum = %f\n", radius, circum);
}
```

- How to print output only upto 2 decimals?

Circumference of circle – formatted output

```
#include<stdio.h>

main() {
    float radius;
    float circum;

    printf("Enter radius : ");
    scanf("%f", &radius);
    circum = 2* (22.0/7) * radius;

    printf ("radius = %5.2f, circum = %5.2f\n", radius, ci
}
```

Output statement

`printf (format-string,` $var_1, var_2, ..., var_n$`)`

Output statement

`printf (format-string,` $var_1, var_2, ..., var_n$`)`

Format string specifies

- How many variables to expect?
- Type of each variable.
- How many columns to use for printing? (width)
- What is the precision? (if applicable)

Output statement

`printf (format-string,` $var_1, var_2, ..., var_n$`)`

Format string specifies

- How many variables to expect?
- Type of each variable.
- How many columns to use for printing? (width)
- What is the precision? (if applicable)
- Common mistakes:
  - comma missing after the double quotes.
  - mismatch in the actual number of variables given and those expected in the format string.

Formatted output

```
printf (‘‘%w.pC", x);
```

Formatted output

```
printf (‘‘%w.pC", x);
```

- w, p and C are place holders, can take different values.
- w: width of the output. (optional)
- p: precision of the output. (optional)
- C: Conversion character.

Formatted output

```
printf (‘‘%w.pC", x);
```

- w, p and C are place holders, can take different values.
- w: width of the output. (optional)
- p: precision of the output. (optional)
- C: Conversion character.
  - d : integer
  - f : float
  - c : character
  - x : hexadecimal
  - o : octal
  - u : unsigned int
  - e : real decimal in exponent form

Input Statement

`scanf (format-string,` $\&var_1, \&var_2, \ldots, \&var_n$`)`

Input Statement

`scanf (format-string,` $\&var_1, \&var_2, \ldots, \&var_n$`)`

Format string specifies

- How many variables to expect?
- Type of each variable.

Input Statement

`scanf (format-string,` $\&var_1, \&var_2, \ldots, \&var_n$`)`

Format string specifies

- How many variables to expect?
- Type of each variable.
- Common mistakes:
  - comma missing after the double quotes.
  - mismatch in the actual number of variables given and those expected in the format string.
  - & missing before the variable.

Learnings so far..

- Variables in C.

Learnings so far..

- Variables in C.
- Data types in C - how they are stored. Why a programmer should be worried.

Learnings so far..

- Variables in C.
- Data types in C - how they are stored. Why a programmer should be worried.
- Arithmetic operators. Precedence of operators.

Learnings so far..

- Variables in C.
- Data types in C - how they are stored. Why a programmer should be worried.
- Arithmetic operators. Precedence of operators.
- Assignment operator "="

Learnings so far..

- Variables in C.
- Data types in C - how they are stored. Why a programmer should be worried.
- Arithmetic operators. Precedence of operators.
- Assignment operator "="
- Formatting the input and output - the `printf` and `scanf`

Learnings so far..

- Variables in C.
- Data types in C - how they are stored. Why a programmer should be worried.
- Arithmetic operators. Precedence of operators.
- Assignment operator "="
- Formatting the input and output - the `printf` and `scanf`
- What is coming up?
  - Compilation and Exection of C-programs.
  - More Programming.