Rajarata University of Sri Lanka
Department of Physical Sciences

COM1407
Computer Programming

LECTURE 09

TYPE CASTING, COMMAND LINE ARGUMENTS AND DEFINING CONSTANTS

T.C IRUGALBANDARA

Objectives

- At the end of this lecture students should be able to;
  - Define type cast and type promotion in C programming language.
  - Define command line arguments in C Programming language.
  - Declare constants according to the C programming.
  - Apply math.h header file for problem solving.
  - Apply taught concepts for writing programs.

Type Casting

- Type cast is an instruction to the compiler to convert one type into another.
- For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'.
- Here, target-type specifies the desired type to convert the specified expression to.
- You can convert the values from one type to another **explicitly** using the **cast operator** as follows:

(type_name) expression

Type Casting (Cont…)

- When a cast involves a narrowing conversion, information might be lost.
- For example, when casting a long into a short, information will be lost if the long's value is greater than the range of a short because its high-order bits are removed.
- When a floating-point value is cast to an integer type, the fractional component will also be lost due to truncation.

Type Casting (Cont...)

```
#include <stdio.h>

int main() {
    int sum = 17, count = 5;
    double mean;
    mean = sum / count;
    printf("Value of mean : %f\n", mean
);
    return 0;
}
```

```
Value of mean : 3.000000
```

Type Casting (Cont…)

```
#include <stdio.h>

int main() {
    int sum = 17, count = 5;
    double mean;
    mean = (double) sum / count;
    printf("Value of mean : %f\n", mean
);
    return 0;
}
```

It should be noted here that the cast operator has precedence over division, so the value of **sum** is first converted to type **double** and finally it gets divided by count yielding a double value.

```
Value of mean : 3.400000
```

Type Promotion/ Automatic Casting

- When one type of data is assigned to another type of variable, an automatic type conversion will take place **if**
  - The two types are compatible. E.g. boolean and char is not compatible.
  - The destination type is larger than source type.
- For example, the int type is always large enough to hold all valid short values, and both int and short are integer types, so an automatic conversion from short to int can be applied. (This is known as integer promotion)

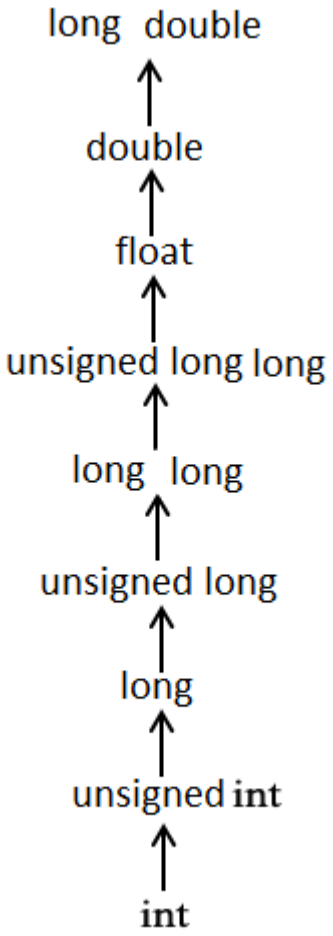
Type Promotion/ Automatic Casting (Cont…)

```
#include <stdio.h>

int main() {
   int  i = 17;
   char c = 'c'; /* ascii value is 99 */
   int sum;

   sum = i + c;
   printf("Value of sum : %d\n", sum );
   return 0;
}
```

Here, the value of sum is 116 because the compiler is doing integer promotion and converting the value of 'c' to ASCII before performing the actual addition operation.

Typical/Usual Arithmetic Conversion

- The **typical arithmetic conversions** are **implicitly** performed to cast their values to a common type.
- The compiler first performs *integer promotion*; if the operands still have different types, then they are converted to the type that appears highest in the given hierarchy.



Typical Arithmetic Conversion (Cont…)

- In any program it is recommended to only perform arithmetic operations on pairs of values of the same type.
- When compatible types are mixed in an assignment, the value of the right side is automatically converted to the type of the left side.
- However, because of C's strict type checking, not all types are compatible, and thus, not all type conversions are **implicitly** allowed.

Typical Arithmetic Conversion (Cont…)

- For widening conversions, the numeric types, including integer and floating-point types, are compatible with each other.
- There are no automatic conversions from the numeric types to char or boolean. Also, char and boolean are not compatible with each other.
- However, an integer literal can be assigned to char

Typical Arithmetic Conversion (Cont…)

```
#include <stdio.h>
int main (void)
{
   float f1 = 123.125, f2;
   int i1, i2 = -150;
   char c = 'a';
   /* floating to integer conversion */
   i1 = f1;
   printf ("%f assigned to an int produces %i\n", f1,
   i1);
   /* integer to floating conversion */
   f1 = i2;
   printf ("%i assigned to a float produces %f\n", i2,
   f1);
```

Typical Arithmetic Conversion (Cont…)

```
/* integer divided by integer */
f1 = i2 / 100;
printf ("%i divided by 100 produces %f\n", i2, f1);
/* integer divided by a float */
f2 = i2 / 100.0;
printf ("%i divided by 100.0 produces %f\n", i2, f2);
/* type cast operator */
f2 = (float) i2 / 100;
printf ("(float) %i divided by 100 produces %f\n", i2,
f2);
return 0;
```

```
123.125000 assigned to an int produces 123
-150 assigned to a float produces -150.000000
-150 divided by 100 produces -1.000000
-150 divided by 100.0 produces -1.500000
(float) -150 divided by 100 produces -1.500000
```

Summary

- Typecasting is also called as type conversion
- It means converting one data type into another.
- Converting smaller data type into a larger one is also called as type promotion.
- 'C' provides an implicit and explicit way of type conversion.
- Implicit type conversion operates automatically when the compatible data type is found.
- Explicit type conversion requires a type casting operator.

Command Line Arguments

- It is possible to pass some values from the command line to your C programs when they are executed.
- These values are called **command line arguments** and many times they are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.
- The command line arguments are handled using main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program.

Command Line Arguments (Cont...)

```
#include <stdio.h>


int main( int argc, char *argv[] )
{
   printf("The first argument supplied is
   %s\n", argv[0]);
   printf("The second arg
   %s\n", argv[1]);
```

When you run the program remember to pass the arguments

```
F:\C\L04>CommandLineArguments.exe test
The first argument supplied is CommandLineArguments.exe
The second argument supplied is test
```

Command Line Arguments (Cont…)

- It should be noted that **argv[0]** holds the name of the program itself and **argv[1]** is a pointer to the first command line argument supplied, and *argv[n] is the last argument.
- If no arguments are supplied, argc will be one, and if you pass one argument then **argc** is set at 2.
- E.g.

```
F:\C\L04>CommandLineArguments.exe
test
```

argv	`CommandLineArguments.exe`	`test`

argc is 2 here

Command Line Arguments (Cont...)

```
#include <stdio.h>


int main( int argc, char
*argv[] )

{

    printf("The first argument
supplied is %s\n", argv[0]);

    printf("The second argument
supplied is %s\n", argv[1]);

    printf("The third argument
supplied is %s\n", argv[2]);

    return 0;

}
```

```
F:\C\L04>CommandLineArguments.exe arg1 arg2
The first argument supplied is
CommandLineArguments.exe
The second argument supplied is arg1
The third argument supplied is arg2

F:\C\L04>CommandLineArguments.exe "arg1"
"arg2"
The first argument supplied is
CommandLineArguments.exe
The second argument supplied is arg1
The third argument supplied is arg2

F:\C\L04>CommandLineArguments.exe "arg1"
arg2
The first argument supplied is
CommandLineArguments.exe
The second argument supplied is arg1
The third argument supplied is arg2
```

Command Line Arguments (Cont...)

```
#include <stdio.h>

int main( int argc, char
*argv[] )
{
    printf("The first
argument supplied is %s\n",
argv[0]);
    printf("The second
argument supplied is %s\n",
argv[1]);
    printf("The third
argument supplied is %s\n",
argv[2]);
    return 0;
}
```

```
F:\C\L04>CommandLineArguments.exe arg1
"arg2"
The first argument supplied is
CommandLineArguments.exe
The second argument supplied is arg1
The third argument supplied is arg2

F:\C\L04>CommandLineArguments.exe arg1, arg2
The first argument supplied is
CommandLineArguments.exe
The second argument supplied is arg1,
The third argument supplied is arg2
```

Defining Constants

- There are two ways to define constant in C programming:
  - Using the const key word.
    - #define come before the program main block.

```
const data_type constant_name = constant_value;
```

```
const float PI = 3.1412;
```

  - Using the #define directive.

```
#define constant_name  constant_value
```

```
#define PI 3.1412
```

Defining Constants (Cont…)

- Write a program to read the radius of a circle and print out the perimeter of it on the screen.

```
#include <stdio.h>
#define PI 3.1412

int main ()
{
     float radius;
     printf ("Enter the radius in mm : \n");
     scanf ("%f",&radius);

     printf("The perimeter is : %.2f", 2*PI*radius);
     return 0;
}
```

Defining Constants (Cont…)

```
#include <stdio.h>
int main (){
   const float PI = 3.1412;
   float radius;
   printf ("Enter the radius in mm : \n");
   scanf ("%f",&radius);

   printf("The perimeter is : %.2f", 2*PI*radius);
   return 0;
}
```

Math Functions

- The **math.h** header defines various mathematical functions.
- All the functions available in this library take **double** as an argument and return **double** as the result.
- You can find documentation via:
  - http://devdocs.io/c/numeric/math

Math Functions (Cont...)

S.N.	Function & Description
1	**double acos(double x)** Returns the arc cosine of x in radians.
2	**double asin(double x)** Returns the arc sine of x in radians.
3	**double atan(double x)** Returns the arc tangent of x in radians.
4	**double atan2(double y, double x)** Returns the arc tangent in radians of y/x based on the signs of both values to determine the correct quadrant.
5	**double cos(double x)** Returns the cosine of a radian angle x.
6	**double cosh(double x)** Returns the hyperbolic cosine of x.
7	**double sin(double x)** Returns the sine of a radian angle x.

Math Functions (Cont...)

8	**double sinh(double x)** Returns the hyperbolic sine of x.
9	**double tanh(double x)** Returns the hyperbolic tangent of x.
10	**double exp(double x)** Returns the value of **e** raised to the xth power.
11	**double frexp(double x, int *exponent)** The returned value is the mantissa and the integer pointed to by exponent is the exponent. The resultant value is x = mantissa * 2 ^ exponent.
12	**double ldexp(double x, int exponent)** Returns **x** multiplied by 2 raised to the power of exponent.
13	**double log(double x)** Returns the natural logarithm (base-e logarithm) of **x**.
14	**double log10(double x)** Returns the common logarithm (base-10 logarithm) of **x**.

Math Functions (Cont...)

15	**double modf(double x, double *integer)** The returned value is the fraction component (part after the decimal), and sets integer to the integer component.
16	**double pow(double x, double y)** Returns x raised to the power of **y**.
17	**double sqrt(double x)** Returns the square root of **x**.
18	**double ceil(double x)** Returns the smallest integer value greater than or equal to **x**.
19	**double fabs(double x)** Returns the absolute value of **x**.
20	**double floor(double x)** Returns the largest integer value less than or equal to **x**.
21	**double fmod(double x, double y)** Returns the remainder of x divided by **y**.

Math Functions (Cont…)

```
#include <stdio.h>
#include <math.h>

int main ()
{
   printf("Value 8.0 ^ 3 = %.2lf\n", pow(8.0, 3));
   printf("Sqrt of 100 = %.2f\n", sqrt(100));
   printf("log (100) = %.2f\n", log(100.0));
   printf("sin (90) = %.2f\n", sin (90.0));
   printf("ceil (10.45) = %.2f\n", ceil(10.45));
   printf("floor (10.45) = %.2f\n", floor(10.45));
   printf("fabs (-10.45) = %.2f\n", fabs(-10.45));
   printf("round (10.45) = %.2f\n", round (10.45));

   return(0);
}
```

Objective Re-cap

- Now you should be able to:
  - Define type cast and type promotion in C programming language.
  - Define command line arguments in C Programming language.
  - Declare constants according to the C programming.
  - Apply math.h header file for problem solving.
  - Apply taught concepts for writing programs.

References

- Appendix A, section 5 - Programming in C, 3rd Edition, Stephen G. Kochan