**Rajarata University of Sri Lanka**

**Faculty of Applied Sciences**

**COM 1407 – Computer Programming**

# Practical – Type Casting in C

**Outline**

- Defining Type Cast
- Defining Command Line Arguments
- Declare Constants
- math.h

## 1 Type Casting

- Converting one data type into another is known as type casting or, type-conversion. For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'.
- **Syntax**

  ```
  (type_name) expression
  ```
- Consider the following example

```
#include <stdio.h>

main() {

int sum = 17, count = 5;
 double mean;

  mean = (double) sum / count;
   printf("Value of mean : %f\n", mean );
}
```

- 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.

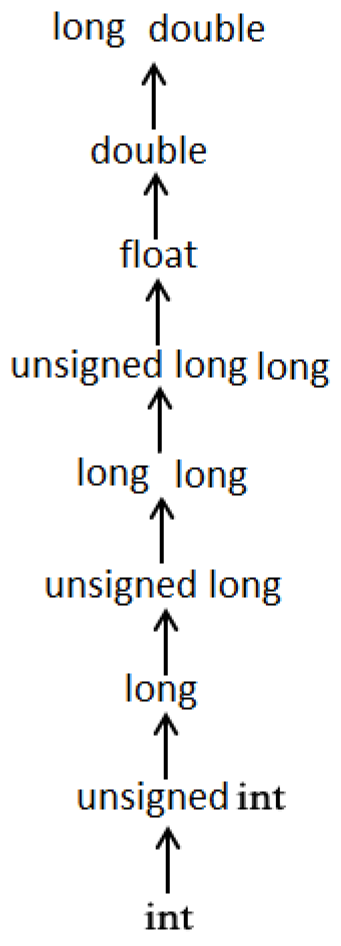
## 1.1 Integer Promotion/Automatic Casting

- Type conversions can be implicit which is performed by the compiler automatically, or it can be specified explicitly through the use of the **cast operator**.
- Integer promotion is the process by which values of integer type "smaller" than **int** or **unsigned int** are converted either to **int** or **unsigned int.**
- Consider the following example

```
#include <stdio.h>

main() {

 int  i = 17;
 char c = 'c'; /* ascii value is 99 */
  int sum;

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

## 1.2 Usual Arithmetic Conversion

- The usual 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 following hierarchy,



## 2 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.
- 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.
- Consider following example

```
#include <stdio.h>

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

  printf("Program name %s\n", argv[0]);

  if( argc == 2 ) {
     printf("The argument supplied is %s\n", argv[1]);
  }
  else if( argc > 2 ) {
     printf("Too many arguments supplied.\n");
   }
   else {
    printf("One argument expected.\n");
 }
}
```

- 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.

## 3 Defining Constants

- Constants refer to fixed values that the program may not alter during its execution.
- Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant.
- There are two simple ways in C to define constants.
    1. Using **#define** preprocessor.

```
#include <stdio.h>

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main() {
   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);
 return 0;
}
```

    2. Using **const** keyword

```
#include <stdio.h>
int main() {
 const int  LENGTH = 10;
const int  WIDTH = 5;
const char NEWLINE = '\n';\
int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);\
printf("%c", NEWLINE);
 return 0;
}
```

## 4 math.h header

- The math.h header defines various mathematical functions and one macro. All the functions available in this library take double as an argument and return double as the result.

| | |
|---|---|
| 1 | double acos(double x)<br><br>Returns the arc cosine of x in radians. |
| 2 | double asin(double x)<br><br>Returns the arc sine of x in radians. |
| 3 | double atan(double x)<br><br>Returns the arc tangent of x in radians. |
| 4 | double atan2(double y, double x)<br><br>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)<br><br>Returns the cosine of a radian angle x. |
| 6 | double cosh(double x)<br><br>Returns the hyperbolic cosine of x. |
| 7 | double sin(double x)<br><br>Returns the sine of a radian angle x. |
| 8 | double sinh(double x)<br><br>Returns the hyperbolic sine of x. |
| 9 | double tanh(double x)<br><br>Returns the hyperbolic tangent of x. |

| | |
|---|---|
| 10 | double exp(double x)<br><br>Returns the value of e raised to the xth power. |
| 11 | double frexp(double x, int *exponent)<br><br>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)<br><br>Returns x multiplied by 2 raised to the power of exponent. |
| 13 | double log(double x)<br><br>Returns the natural logarithm (base-e logarithm) of x. |
| 14 | double log10(double x)<br><br>Returns the common logarithm (base-10 logarithm) of x. |
| 15 | double modf(double x, double *integer)<br><br>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)<br><br>Returns x raised to the power of y. |
| 17 | double sqrt(double x)<br><br>Returns the square root of x. |
| 18 | double ceil(double x)<br><br>Returns the smallest integer value greater than or equal to x. |
| 19 | double fabs(double x)<br><br>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)<br>Returns the remainder of x divided by y. |