Data Types & Arithmetic Expressions

1. Objective

- To be able to list, describe, and use the C basic data types.
- To be able to create and use variables and constants.
- To be able to use simple input and output statements.
- Learn about type conversion.

2. Data Types

- A type defines by the following:
  - A set of **values**
  - A set of **operations**

- C offers three basic data types:
  - **Integers** defined with the keyword **int**
  - **Characters** defined with the keyword **char**
  - **Real or floating** point numbers defined with the keywords **float** or **double**.

3. Integers

- positive or negative whole numbers
- Thre are three types of integers:
  - “int”,
  - “short int” (which can be abbreviated “short”)
  - “long int” (which can be abbreviated “long”).

- Example:

```
int myVarInt;
int myDistance;

myVarInt = 16;
myDistance = 4;
printf("myVarInt:%d\n", myVarInt);
printf("myVarInt:%i\n", myVarInt);

printf("myDistance:%d\n", myDistance);
printf("myDistance:%i\n", myDistance);
```

4. Real numbers

- There is an infinite number of real numbers, but they are represented with a finite number of bits in the computer.
- There are two main types:
  - “float”

    - "double".
- Real numbers defined with "double" are represented with a size double of those declared as "float".
    - The difference is the amount of precision used to represent the numbers internally.
    - Example:

```
float a = 3.5;
double b = -5.4e-12;
long double c = 3.54e320;
```

    - Print a real number:

```
double myVard;
float myVarf;
long double myVarld = 3.54e30;

myVard = 9.72;
myVarf = 9.72;
myVarld = 3.54e30;
printf("This is a double:%f\n", myVard);
printf("This is a double:%e\n", myVard);
printf("This is a float:%f\n", myVarf);
printf("This is a float:%e\n", myVarf);
printf("This is a long double:%Le\n", myVarld);
```

5. Characters and strings

- **Character**:
  - The variables of type character are declared as “char”.
  - To refer to a character, the symbol must be surrounded by simple quotes: 'M'.
  - Characters are internally represented as numbers and the C language allows arithmetic operations with them such as 'M' + 25.
  - Print a character:

```
char myChar;
myChar = 'A';
printf("This is a character:%c\n", myChar);

myChar = 'A'+1;
printf("This is a character:%c\n", myChar);

myChar = 'A'+2;
printf("This is a character:%c\n", myChar);
```

- **String**:
  - The strings are represented as tables of “char”.
  - The library functions to manipulate strings all assume that the last byte of the chain has value zero.
  - The strings are written in the program surrounded by double quotes and contain the value zero at the end.
  - Print a character:

```
char myText1 [30]="I am printing a string";
char myText2 [20]="I am printing a string";
char myText3 [10]="I am printing a string";
printf("mText1:%s\n", myText1);
printf("mText2:%s\n", myText2);
printf("mText3:%s\n", myText3);
```

6. Basic Data Types Sizes: In Class

- The C programming languages does not define a fixed size for the basic data types.
- The size of each data type depends on the implementation.

Type	Bits	Sign	Range
Char	8	Unsigned	0 .. 255
signed char	8	Signed	-128 .. 127
unsigned short	16	Unsigned	0 .. 65,535
short	16	Signed	-32,768 .. 32,767
unsigned int	32	Unsigned	0 .. 4,294,967,295
Int	16 OR 32	Signed	-32,768 to 32,767 OR -2,147,483,648 .. 2,147,483,647
unsigned long long	64	Unsigned	0 .. 18,446,744,073,709,551,615
long long	64	Signed	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807

- Example:

```
//gcc 5.4.0

#include  <stdio.h>

// size of basic data types in C
int main(void)
{
   int answer;
   short myFirst = 1;
   long mySecond = 2;
   float myThird = 3.0;
   double myFourth = 4.4;
   long double myFifth = 5.54;
   char myCharacter = 'p';


  /* The size of various types */
  printf("The size of int         %zu\n", sizeof(answer));
  printf("The size of short       %zu\n", sizeof(myFirst));
  printf("The size of long        %zu\n", sizeof(mySecond));
  printf("The size of float       %zu\n", sizeof(myThird));
  printf("The size of double      %zu\n", sizeof(myFourth));
  printf("The size of long double %zu\n", sizeof(myFifth));
  printf("The size of char        %zu\n", sizeof(myCharacter));

  return 0;
}
```

7. Constant Variables

- Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**:
  - **Integer Literals**: can any number. e.g., integers, reals, etc.
  - **Character Literals**: any character and are enclosed in single quotes, e.g., 'x' .
  - **String Literals**: any sequence of character and are enclosed in double quotes "".

- A good practice is to minimize the use of literal numbers in code.
- One reason is to improve code readability.
- A common convention, or good practice, is to name constant variables using upper case letters, to make constant variables clearly visible in code.
- There are two simple ways in C to define constants:
  - Using **#define** preprocessor.
  - Using **const** keyword.

- **Examples:**

```
const int MAXRATE = 10; /*int constant*/
const float PI = 3.14; /*Real constant*/
const char MYCHARACTER = 'A'; /*char constant*/
const char MYAREA[10] = "Tysons Corner"; /*string constant*/
const double SPEED_OF_SOUND   = 761.207; // Miles/hour (sea level)
const double SECONDS_PER_HOUR = 3600.0;  // Secs/hour
#define NEWLINE '\n'
```

8. Questions/Practice

1. The number of cars in a parking lot.

   double
   int

2. The current temperature in Celsius.

   double
   int

3. A person's height in centimeters.

   double
   int

4. The number of hairs on a person's head.

   double
   int

5. The average number of kids per household.

   double
   int

9. Input Statement

- In C programming language, scanf() function is used to read character, string, numeric data from keyboard.
- scanf() is a predefined function in "stdio.h" header file.
- Syntax:

```
scanf("format specifiers",&value1,&value2,.....);
```

- Example 1:

```
//gcc 5.4.0
#include  <stdio.h>
int main(void)
{
	int a;
	float b;
	scanf("%d%f",&a,&b);
	printf("a:%d ---- b:%f", a, b);
}
```

- Example 2:

```
//gcc 5.4.0
#include  <stdio.h>
int main(void)
{
  char ch;
  char str[100];
  printf("Enter any character \n");
  scanf("%c", &ch);
  printf("Entered character is %c \n", ch);
  printf("Enter any string ( upto 100 character ) \n");
  scanf("%s", &str);
  printf("Entered string is %s \n", str);
}
```

- Example 3:

```
//gcc 5.4.0

#include  <stdio.h>

int main(void)

{

   int x , y;

   printf("Enter a value for x:");

   scanf("%d\n", &x);

   printf("%d\n", x);

   printf("Enter a value for y:");

   scanf("%d\n", &y);
```

```
    printf("%d\n", y);
    printf("The sum of x and y is: %d\n", x+y);
}
```

- Format specifiers for printf() and scanf() statements.

Format specifier	Data type	Notes
%c	char	Prints or reads a single ASCII character
%d	int	Prints or reads a decimal integer values.
%hd	short	Prints or reads a short signed integer.
%ld	long	Prints or reads a long signed integer.
%lld	long long	Prints or reads a long long signed integer.
%u	unsigned int	Prints or reads an unsigned integer.
%hu	unsigned short	Prints or reads an unsigned short integer.
%lu	unsigned long	Prints or reads an unsigned long integer.
%llu	unsigned long long	Prints or reads an unsigned long long integer.
%f	float	Prints or reads a float floating-point value.
%lf	double	Prints or reads a double floating-point value (lf stands for long float).

%s	string	printf() will print the contents of a string up to the null character. scanf() will read a string of characters from the user input until a whitespace character (a space, tab, or newline) is reached.
%%		Prints the % character.

10. Arithmetic Expressions

- As in most languages, C programs specify computation in the form of arithmetic expressions that closely resemble expressions in mathematics.
- The most common operators in C are the ones that specify arithmetic computation:

Arithmetic operator	Description
+	addition
-	subtraction
*	multiplication
/	division
%	modulo (remainder)

- Binary Operators:
  - Operators in C usually appear between two subexpressions, which are called its **operands**. **Operators** that take two operands are called binary operators:

    Operand operator Operand

    A + B

  - Unary Operator:

- The - operator can also appear as a **unary operator**, as in the expression -x, which denotes the negative of x.

- **Precedence rules for arithmetic operators**:

Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In 2 * (A + 1), A + 1 is computed first, with the result then multiplied by 2.
unary -	- used as a negative (unary minus) is next	In 2 * -A, -A is computed first, with the result then multiplied by 2.
* / %	Next to be evaluated are *, /, and %, having equal precedence.	
+ -	Finally come + and - with equal precedence.	In B = 3 + 2 * A, 2 * A is evaluated first, with the result then added to 3, because * has higher precedence than +.
left-to-right	If more than one operator of equal precedence could be evaluated, evaluation occurs left to right.	In B = A * 2 / 3, A * 2 is first evaluated, with the result then divided by 3.

- Example: In Class
  - Compute the solutions of a quadratic equation

```
//gcc 5.4.0

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

int main(void)
{
   // y = ax**2+bx+c
   //Compute quadratic formula
   // [-b+/-squareroot(b**2-4ac)]/2*a

   int a = 1;
   int b = 0;
   int c = -1;

   double discriminant = powf(b,2) - 4*a*c;
   double x =   sqrt(discriminant);

   double solution1 = (-b + x )/(2*a);
   double solution2 = (-b - x )/(2*a);

   printf("solution1: %f\n", solution1);
   printf("solution2: %f\n", solution2);

   return 0;
}
```

- An example of Modulo operator:

```
//gcc 5.4.0

#include  <stdio.h>

int main(void)
{
   int x = 89;
   printf("The reminder of the division of %d by 10 = %d\n",x,  x % 10);

   x = 9;
   printf("The reminder of the division of %d by 10 = %d\n",x,  x % 10);

   x = 20;
   printf("The reminder of the division of %d by 10 = %d\n",x,  x % 10);

   //How to find out if a number is even
   x = 3488;
   printf("The reminder of the division of %d by 2 = %d\n",x,  x % 2);
   x = 3489;
   printf("The reminder of the division of %d by 2 = %d\n",x,  x % 2);

   return 0;
}
```

- Random Numbers

```
//gcc 5.4.0

#include <stdio.h>
#include <stdlib.h>   // Enables use of rand()

int main(void) {

   int myRand;
   printf("Four rolls of a dice...\n");

   // rand() % 6 yields 0, 1, 2, 3, 4, or 5
   // so + 1 makes that 1, 2, 3, 4, 5, or 6

   myRand = rand();
   printf("Random Number = %d\n", myRand);
   printf("%d\n", ((myRand % 6) + 1));
   myRand = rand();
   printf("Random Number = %d\n", myRand);
   printf("%d\n", ((myRand % 6) + 1));
   myRand = rand();
   printf("Random Number = %d\n", myRand);
   printf("%d\n", ((myRand % 6) + 1));
   myRand = rand();
   printf("Random Number = %d\n", myRand);
   printf("%d\n", ((myRand % 6) + 1));

  return 0;
}
```

11. Questions/Practice

- Write a program that reads two integers and print their sum.

- Write a program that computes the area of a circle. Given the radius of a circle, the area is:

$$\pi * R^2$$

Where R is the radium and $\pi = 3.14$.

- Write a program that computes the volume of a sphere. Given the radius of the sphere, the volume is:

$$(4.0 / 3.0)\ \pi\ R^3$$

Where R is the radium and $\pi = 3.14$.

12. Shorthand operators

- A shorthand operator is a shorter way to express an expression.

- Shorthand operators +=, -=, *=, /= and *=

  - Frequent expressions:

    x is a variable in the program

Operator	Name	Example	Equivalent construct
+=	Addition assignment	x += 4;	x = x + 4;
-=	Subtraction assignment	x -= 4;	x = x - 4;
*=	Multiplication assignment	x *= 4;	x = x * 4;
/=	Division assignment	x /= 4;	x = x / 4;
%=	Remainder assignment	x %= 4;	x = x % 4;

- Special Statements: Increment and decrement operators:

Operator	Name	Example	Equivalent construct
++	Increment	x++;	x = x + 1;
--	Decrement	x--;	x = x - 1;

- Example:

```
//gcc 5.4.0

#include  <stdio.h>

int main(void)
{
   int x;
   int y;
   x = 7;
       printf("x=%d\n",x);
   x++;
   printf("x before incrementing x=%d\n\n",x);

   x = 7;
       printf("x=%d\n",x);
       ++x;
   printf("x after incrementing x=%d\n\n",x);


   x = 7;
       printf("x=%d\n",x);
   x--;
   printf("x before decrementing x=%d\n\n",x);

   x = 7;
       printf("x=%d\n",x);
   --x;
   printf("x after decrementing x=%d\n\n",x);
```

```
    y = x;
        printf("y=%d\n",y);
    y = x++;
    printf("y before incrementing x=%d\n\n",y);

    y = x;
        printf("y=%d\n",y);
    y = ++x;
    printf("y after incrementing x=%d\n\n",y);

    y = x;
        printf("y=%d\n",y);
    y = x--;
    printf("y before decrementing c=%d\n\n",y);

    y = x;
        printf("y=%d\n",y);
    y = --x;
    printf("y after decrementing x=%d\n",y);

    return 0;
}

//gcc 5.4.0

#include  <stdio.h>

int main(void)
{
```

```
    int x;
    int y;
    x = 7;
    y = 8;
        printf("x=%d\n",x);
        printf("y=%d\n",y);
    x *=y;
    printf("x=%d\n\n",x);

    x = 7;
    y = 8;
    x *=--y;
    printf("x=%d\n\n",x);

    x = 7;
    y = 8;
    x *=y--;
    printf("x=%d\n\n",x);


    return 0;
}
```

13. Questions/Practice

1. numAtoms is initially 7. What is numAtoms after: numAtoms += 5?

_____

2. numAtoms is initially 7. What is numAtoms after: numAtoms *= 2?

_____

3. Rewrite the statement using a compound operator, or type “Not possible”

```
carCount = carCount / 2;
```

_____

4. Rewrite the statement using a compound operator, or type “Not possible”

```
numItems = boxCount + 1;
```

_____

5. A drink costs 2 dollars. A taco costs 3 dollars. Given the number of each, compute total cost and assign to totalCost. Ex: 4 drinks and 6 tacos yields totalCost of 26.

```
#include <stdio.h>

int main(void) {
   int numDrinks = 0;
   int numTacos  = 0;
   int totalCost = 0;

   numDrinks = 4;
   numTacos  = 6;

   /* Your solution goes here  */

   printf("Total cost: %d\n", totalCost);

   return 0;
}
```

14. Type conversions

- A **type conversion** (also known as type **casting**, and type **coercion**) is a conversion of one data type to another, such as an int to a double.

- It is needed when the types of an expression are not compatible:

```
//gcc 5.4.0

#include  <stdio.h>

const int MULT = 5;
int main(void)
{
   int x;
   double y;
   float z = 3.597;
   x = MULT * z;
   y = MULT * z;
   printf("x=%d\n", x);
   printf("y=%f\n", y);
   return 0;
}
```

- There are two types of type conversions:
    - **Implicit conversion (Also known as coercion):** When the compiler automatically performs several common conversions between int and double types.
    - **Explicit conversion (Also known as casting):** When the user decides the type of the conversion (the desired type)

- For assignment =, the right side type is converted to the left side type.
    - int-to-double conversion is straightforward
        - 25 becomes 25.0.
    - double-to-int conversion just drops the fraction:
        - 4.9 becomes 4.

- Arithmetic Expressions:
  - Conversions are implicitly performed to cast their values to a common type, if the user does not specify any casting.
  - The compiler user the hierarchy: