Variables, Expressions and Assignments

We first write a program "simpint.c" that calculates the total amount accrued on a principal of Rs. 100, 000 invested at a simple interest rate of 5% per annum for 5 years. For this we use the well known formula

$$\text{Total amount} = P\,(1 + R\,T),$$

where $P = 100000$, $R = 5\% = 0.05$, and $T = 5$ respectively denote the principal, interest rate, and number of years.

Our program will use variables that can store integer values and real values. These variables must be declared or named in the beginning of the program. These variable names are also called identifiers. An identifier can be defined using a sequence of letters, digits, and underscores. But it should not start with a digit. In addition, you are not allowed to use special characters. For example '+' is a special character that is used for arithmetic addition. Hence, an identifier should not have '+' as a character. Thus, "`int a2+b;`" is not correct. Also, as we have mentioned earlier that function "printf()" is already written and compiled in C compiler. Similarly, later we will use trigonometric functions like "sin()", "cos()", exponential function "exp()", etc., that are already written and compiled in C compiler. These functions are written with identifiers starting with an underscore. To avoid clash with identifiers used in pre-compiled C functions, we will not use identifiers starting with an underscore (though it is legal).

```
/* Calculate total amount accrued in 5 years when rate
 of interest is 5% and principal amount is Rs. 100000.00*/
#include<stdio.h>
int main()
{
 int years;
 double princp,rate,total;

 years=5;  //no of years
 princp=100000.00; //principal amount
 rate=0.05;  //rate of interest

 total=princp*(1+rate*years);

 printf("Total amount accrued in %d years is Rs. %lf\n",years,total);
 return 0;
}
```

Listing 4: C program "simpint.c"

Compiling with `$ gcc simpint.c ↩` and running the default executable with `$ ./a.out ↩` produce the following output

```
Total amount accrued in 5 years is Rs. 125000.000000
```

Analysis of "simpint.c" program

☐
```
/* Calculate total amount accrued in 5 years when rate
 of interest is 5% and principal amount is Rs. 100000.00*/
```

This is a multi-line comment that has already been discussed.

☐ `int years;`

This is a declaration for the identifier "years". A declaration ends with a semicolon just like a statement. `int` is a keyword and the variables following this are of "int" type: taking integer values. `int` is one of the fundamental types in C language. Note that in C, the data type `int` can be modified using keywords like short, long, signed, and unsigned to control its size (in machine representation) and the range of values it can store. We will discuss these in detail later. However, `int` is widely used and sufficient for most purposes.

☐ `double princp,rate,total;`

This is again a declaration that informs the compiler that the identifiers "princp", "rate" and "total" are of data type "double" taking real values. This is again one of the fundamental types in C. Real values in C can also be declared using `float` and `long double`. The main differences among `float`, `double` and `long double` are their size (in machine representation), which determines their precision and range. Roughly, `double` has more accuracy than `float` and most of our code will be based on `double`. We will discuss their machine representation in detail later. As far as possible, we try to match the identifier to the problem variable. For example, we use "rate" for "interest rate".

☐
```
years=5;  //no of years
princp=100000.00; //principal amount
rate=0.05;  //rate of interest
```

These are assignment statements. We have also used single-line comment to give a brief description of each identifier. The number 5 is an integer constant, whereas 100000.00 and 0.05 are real constants. The equal sign `=` is an assignment operator. The value 5 is assigned to the variable `years`. Similarly, 100000.00 and 0.05 are assigned to the identifiers `princp` and `rate`.

- `total=princp*(1+rate*years);`

  This is again an assignment statement. The value of the expression `princp*(1+rate*years)` on the right of equal sign `=` is evaluated and assigned to to the variable `total`. The operators `*` and `+` stand for arithmetic multiplication and addition respectively. Operations inside the parentheses are performed first. Since multiplication has higher precedence than addition, the value of the sub-expression "`rate*years`" is calculated first. This value is then added to `1` to produce a value that is then multiplied by `princp`. This final value is then assigned to `total`. When an operation involves both integer and real operands (such as `rate*years`), C performs an implicit type conversion to ensure consistent data types during the calculation. Usually, integer component, in an operation involving integer and real, will be promoted to real. Here, `years` is converted to double and `rate*years` becomes multiplication of two doubles. Similarly, "`1`" in `1+rate*years` is first converted to double and then the addition is carried out.

- `printf("Total amount accrued in %d years is Rs. %lf\n",years,total);`

  As discussed earlier, this statement call "printf()" function. The function "printf()" can have variable number of arguments. In this code, it has three arguments, namely, `"Total amount accrued in %d years is Rs. %lf\n"`, `years`, and `total`. The first argument `"Total amount accrued in %d years is Rs. %lf\n"` is called control string. There are two conversion specifications (or formats), namely `%d` and `%lf`. The formats (if any) in a control string are matched with remaining arguments of the "printf()" function. The first format `%d` is matched with the variable `years` and thus it print the value of `years` as an integer value. The second format `%lf` is matched with the variable `total` and thus it print the value of `total` as a double value. Note that the double value is printed with six-decimal places by default. However, later we will show how to control the number of decimal places in the output.

Remark about identifiers: We have seen in "simpint.c" that `years` is an integer variable declared using `int`. Here, `int` is a keyword and similarly `double` is also a keyword. Thus, `int` and `double` cannot be used as names of variables. There is a list of keywords in C (please refer to any reference book) and these cannot be used as identifiers. Similarly, other names known to C compiler can not be used as identifiers. For example, you are not allowed to use `printf` as name of a variable since it is a function in standard library.

Remark about conversion rules: Consider the following program:

```
//Illustration of conversion rules
#include<stdio.h>
int main()
{
 int p,q;
```

```
 double x,y,z;
 x=3/2;           //integer division
 y=3/2.0;         //real division
 p=3;
 q=2;
 z=p/(double)q;   //real division
 printf("x=%lf  y=%lf  z=%lf\n",x,y,z);
 x=3;
 y=13;
 z=14;
 p=y/x;          //real division
 q=z/x;          //real division
 printf("p=%d  q=%d\n",p,q);
 return 0;
}
```

Listing 5: C program "conv.c"

If we compile and run the program in my macOS, the following output is observed:

```
x=1.000000  y=1.500000  z=1.500000
p=4  q=4
```

Note that if our intention is to get 1.5 as the answer, then "`x=3/2;`" does not serve our purpose. This is due to the fact that the expression to the right of `=` is evaluated first irrespective of the variable to the left. Since both the operands "3" and "2" are integers, the division will be integer division leading to "1" as the result. Since the left of `=` is a double, the integer "1" is converted to double and assigned to x. If one of the operand is integer and the other is real, then the integer is converted to real and real division performed. Thus, "`y=3/2.0;`" gives the correct result. Similarly, if we have two identifiers "p" and "q" representing integers and we want real division, then we convert one of the identifier to real. In the expression "`z=p/(double)q;`", the integer variable "q" is cast into a real variable by "`(double) q`". Note that type casting can be applied to "p" too.

Now the integer constant "3" in the expression "`x=3;`" is converted to real and assigned real to variable "x". Similar is the case for "y" and "z". Thus, we have real division in both the expressions "`p=y/x;`" and "`q=z/x;`". In the first case the integer part of the result "13.0/3.0=4.33333$\cdots$" is assigned to "p". In the later case, again the integer part of the result "14.0/3.0=4.6666$\cdots$ is assigned to "q". The point to note is that the result is *not* rounded to nearest integer, only the decimal or fractional part is discarded.

Return to Lecture Topics