The First C Program

- Rules applicable to all C programs:

- Each instruction in a C program is written as a separate statement.
- The statements in a program must appear in the same order in which we wish them to be executed;
- Blank spaces may be inserted between two words to improve the readability of the statement. However, no blank spaces are allowed within a variable, constant or keyword.
- All statements are entered in small case letters .
- C has no specific rules for the position at which a statement is to be written.
- Every C statement must end with a **;**. Thus **;** acts as a statement terminator.

The First C Program

To calculate simple interest for a set of values representing principle, number of years and rate of interest

Formula for computing simple interest

```
si = p * n * r / 100 ;
```

Calculate Simple Interest

```
/* Calculation of simple interest */
/* Date:20/10/2021 */
int main( )
{
   int p, n ;
   float r, si ;
   p = 1000 ;
   n = 3 ;
   r = 8.5 ;
   /* formula for simple interest */
   si = p * n * r / 100 ;
   printf ( "%f" , si ) ;
}
```

Comments

```
/* Calculation of simple interest */
/* Date: 20/10/2021 */

int main( )
{
  int p, n ;
  float r, si ;
  p = 1000 ;
  n = 3 ;
  r = 8.5 ;
  /* formula for simple interest */
  si = p * n * r / 100 ;
  printf ( "%f" , si ) ;
}
```

- Comments should be enclosed within /* */.
- Mentioning the purpose of the statement using a comment.
- Any number of comments can be written at any place in the program.

- The normal language rules do not apply to text written within **/* .. */**.

- Comments cannot be nested. For example,

/* Cal of SI /* Author sam date 01/01/2002 */ */ is invalid.

- A comment can be split over more than one line, as in,

```
/* This is
a jazzy
comment */
```

- ANSI C permits single line comments to be written as

// Calculation of Simple Interest

main( )

```
/* Calculation of simple interest */
/* Date: 20/10/2021 */

int main( )
{
   int p, n ;
   float r, si ;
   p = 1000 ;
   n = 3 ;
   r = 8.5 ;
   /* formula for simple interest */
   si = p * n * r / 100 ;
   printf ( "%f" , si ) ;
  return 0;
}
```

- main is a function.
- Every c program must have main().
- All statements that belong to main() are enclosed within a pair of braces.

```
main( )
{
statement 1 ;
statement 2 ;
Statement 3;
}
```

- main( ) function always return an integer value.

variables

```
/* Calculation of simple interest */
/* Date: 20/10/2021 */

int main( )
{
  int p, n ;
  float r, si ;
  p = 1000 ;
  n = 3 ;
  r = 8.5 ;
  /* formula for simple interest */
  si = p * n * r / 100 ;
  printf ( "%f" , si ) ;
return 0;
}
```

- Any variable used in the program must be declared before being used. Example:-

```
int p, n ;
float r, si ;
```

- Any C statement always ends with a ;
- Example,

```
float r, si ;
r = 8.5 ;
```

- si = p * n * r / 100 ;

printf()

```
/* Calculation of simple interest */
/* Date: 20/10/2021 */

int main( )
{
   int p, n ;
   float r, si ;
   p = 1000 ;
   n = 3 ;
   r = 8.5 ;
   /* formula for simple interest */
   si = p * n * r / 100 ;
   printf ( "%f" , si ) ;
return 0;
}
```

- All output to screen is achieved using library function like printf( )
- Once the value of **si** is calculated it needs to be displayed on the screen.
- The general form of **printf( )** function is,

  printf ("<format string>", <list of variables>) ;
- <format string> can contain,

  %f for printing real values

  %d for printing integer values

  %c for printing character values

printf()

- Following are some examples of usage of **printf( )** function:

  a) printf ("%f", si) ;

  b) printf ("%d %d %f %f", p, n, r, si) ;

  c) printf ("Simple interest = Rs. %f", si) ;

  d) printf ("Prin = %d \nRate = %f", p, r) ;

  (\n is called newline and it takes the curson to next line)

  e) printf ("%d %d %d %d", 3, 3 + 2, c, a + b * c – d) ;

Compilation & Execution

- Type the program and instruct the machine to execute it.
- Editor - To type C program
- Compiler -Typed program needs to be converted to machine language
- To improve programming efficiency
  - Preprocessor
  - Linker
  - Debugger
- Integrated Development Environment (IDE)

Elements of a C Program

- **A ‘C’ development environment includes**
  - ***System libraries* and *headers***: a set of standard libraries and their header files.
  - ***Application Source***: application source program and header files
  - ***Compiler***: converts **source** to object code for a specific platform
  - ***Linker***: resolves external references and produces the executable module

Compiling C programs requires you to work with five kinds of files:

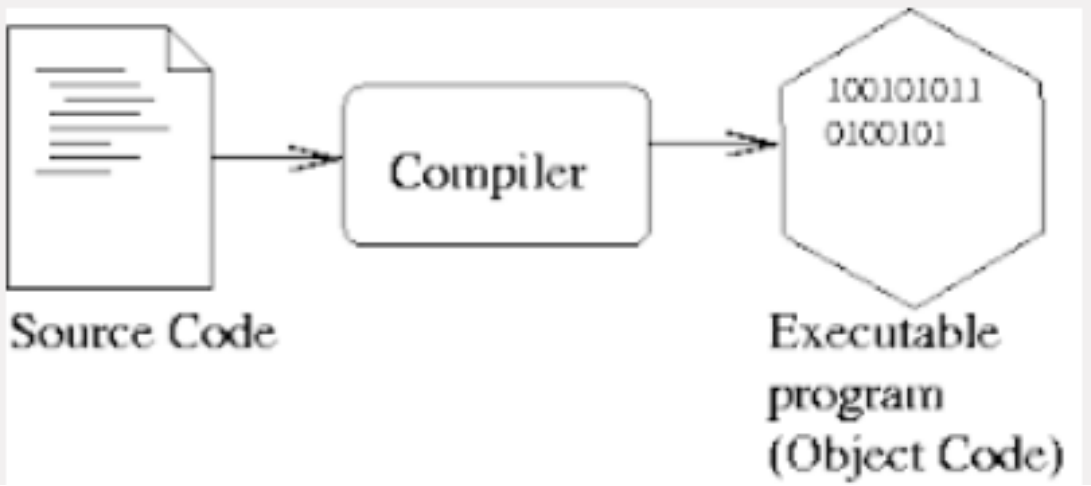
1.Source files: These files contain function definitions, and have names which end in .c by convention. Note: .cc and .cpp are C++ files; not C files.
e.g., foo.c
2.Header files: These files contain function prototypes and various pre-processor statements. They are used to allow source code files to access externally-defined functions. Header files end in .h by convention.
e.g., foo.h
3.Object files: These files are produced as the output of the compiler. They consist of function definitions in binary form, but they are not executable by themselves. Object files end in .o by convention, although on some operating systems (e.g. Windows, MS-DOS), they often end in .obj.
e.g., foo.o, foo.obj
4.Binary executables: These are produced as the output of a program called a "linker". The linker links together a number of object files to produce a binary file which can be directly executed. Binary generally end in .exe on Windows.
e.g., foo foo.exe
5.Libraries: A library is a compiled binary but is not in itself an executable (i.e., there is no main() function in a library). A library contains functions that may be used by more than one program. A library should ship with header files which contain prototypes for all functions in the library; these header files should be referenced. (e.g; #include <library.h>) in any source file that uses the library. The linker then needs to be referred to the library so the program can successfully compiled. There are two types of libraries: static and dynamic.

Preprocessor

- The C preprocessor is a *macro processor* that is used automatically by the C compiler to transform your program before actual compilation.
- Three transformations that the preprocessor always makes on all the input it receives, even in the absence of directives.
  - All C comments are replaced with single spaces.
  - Backslash-Newline sequences are deleted, no matter where. This feature allows you to break long lines for cosmetic purposes without changing their meaning.
  - Predefined macro names are replaced with their expansions

Compiler

- A compiler is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language).
- Typically, from high level source code to low level machine code or object code.



Linker

- Linker is a computer program that links and merges various object files together in order to make an executable file.

- The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded making the program instruction to have absolute reference.

Loader

- Loader is a part of operating system and is responsible for loading executable files into memory and execute them.
- It calculates the size of a program (instructions and data) and create memory space for it. It initializes various registers to initiate execution.

Receiving Input

PREVIOUS CODE

```
/* Calculation of simple interest */
/* Date: 9/08/2020 */

int main( )
{
   int p, n ;
   float r, si ;
   p = 1000 ;
   n = 3 ;
   r = 8.5 ;
   /* formula for simple interest */
   si = p * n * r / 100 ;
   printf ( "%f" , si ) ;
return 0;
}
```

MODIFIED CODE

```
/* Calculation of simple interest */
/* Date: 9/08/2020 */

int main( )
{
   int p, n ;
   float r, si ;
    printf ( "Enter values of p, n, r" ) ;
    scanf ( "%d %d %f", &p, &n, &r ) ;
   /* formula for simple interest */
   si = p * n * r / 100 ;
   printf ( "%f" , si ) ;
return 0;
}
```

Receiving Input

```
/* Calculation of simple interest */
/* Date: 9/08/2020 */

int main( )
{
   int p, n ;
   float r, si ;
    printf ( "Enter values of p, n, r" ) ;
    scanf ( "%d %d %f", &p, &n, &r ) ;
   /* formula for simple interest */
   si = p * n * r / 100 ;
   printf ( "%f" , si ) ;
Return 0;
}
```

- The first **printf( )** outputs the message 'Enter values of p, n, r' on the screen.
- 'Address of' operator.
- It gives the location number used by the variable in memory
- A blank, a tab or a new line must separate the values supplied to **scanf( )**.

Ex.: The three values separated by blank

```
1000 5 15.5
```

Ex.: The three values separated by tab.

```
1000   5    15.5
```

Ex.: The three values separated by newline.

```
1000
 5
15.5
```

Program

Ramesh’s basic salary is input through the keyboard. His dearness allowance is 40% of basic salary, and house rent allowance is 20% of basic salary. Write a program to calculate his gross salary?

Ramesh's basic salary is input through the keyboard. His dearness allowance is 40% of basic salary, and house rent allowance is 20% of basic salary. Write a program to calculate his gross salary?

Algorithm

Step 1: Input BS.

/* BS is a variable that represents Basic Salary */

Step 2: Compute DA

/* DA is a variable that represents Dearness Allowance */

DA = 40 * BS /100 ;

Step 3: Compute RA

/* RA is a variable that represents Rent Allowance */

RA = 20 * BS /100;

Step 4: Calculate GS

/* GS is a variable that represents Gross Salary */

GS = BS + DA + RA

Step 5: PRINT GS

Step 6: End