Programming Language Lab

Bibhas Adhikari

IIT Kharagpur

September 11, 2020

GCC COMPILER

1. The gcc compiler is an ANSI (American National Standard for Information Systems. ANSI's common goal is to provide computing standards for people who use information systems) standard C compiler.
2. A C program goes through a lot of steps prior to becoming a running or executing program.
3. The gcc compiler performs a number of tasks for you. Most notable are the following:
   - Preprocesses the program code and looks for various directives
   - Generates error codes and messages, if applicable.
   - Compiles program code into an object code and stores it temporarily on disk.
   - Links any necessary library to the object code and creates an executable file and stores it on disk.
4. Use the *.c* extension when creating and saving C programs. This extension is the standard naming convention for programs created in C.

How to debug C programs

fflush()

The *fflush()* function ensures that the print statement is sent to your screen immediately, and we should use it if we are using *printf()* for debugging purposes.

1. forget to insert a beginning or a corresponding ending program block identifier
2. missing statement terminators (semicolons)
3. invalid preprocessor directive, such as misspelling a library name
4. invalid comment blocks can generate compile errors

PRIMARY DATA TYPES

Memory concepts

- A computer's memory is somewhat like a human's, in that a computer has both short-term and long-term memory
- A computer's long-term memory is called nonvolatile memory and is generally associated with mass storage devices, such as hard drives, large disk arrays, optical storage (CD/DVD), and of course portable storage devices such as USB flash or key drives
- A computer's short-term, or volatile memory loses its data when power is removed from the computer. It is commonly referred to as RAM (random access memory)
- RAM is comprised of fixed-size cells with each cell number referenced through an address
- Programmers commonly reference memory cells through the use of variables. There are many types of variables, depending on the programming language, but all variables share similar characteristics

Common variable characteristics

Variable	Attribute Description
Name	The name of the variable used to reference data in program code
Type	The data type of the variable (number, character, and so on)
Value	The data value assigned to the memory location
Address	The address assigned to a variable, which points to a memory cell location

Data types

1. There are many data types such as numbers, dates, strings, Boolean, arrays, objects, and data structures
2. First we will concentrate on the following primary data types:
   - Integers : Integer data types hold a maximum of four bytes of information, a single *int* declaration statement with each variable name separated by commas, as:
     int x, y, z;
   - Floating-point numbers: all numbers, including signed and unsigned decimal and fractional numbers. Use the keyword *float* to declare floating-point numbers, as:
     float operand1;
     float operand2;
     float result;
   - Characters : representations of integer values known as character codes.there are a total of 128 common character codes (0 through 127), which make up the most commonly used characters of a keyboard, organized through ASCII

Example: Common variable attributes

Variable name	Value	Type	Memory Address
operand1	29	integer	FFF4
Result	102.5	float	FHH6
Initial	M	Character	FHF2 height

Initializing variables

1. When variables are first declared, the program assigns the variable name (address pointer) to an available memory location
2. It is never safe to assume that the newly assigned variable location is empty. It's possible that the memory location contains previously used data (or garbage).
3. To prevent unwanted data from appearing in your newly created variables, initialize the new variables, as:

```
/* Declare variables */
int x;
char firstInitial;

/* Initialize variables */
x = 0;
firstInitial = '\0';
```

Conversion specifier used with *printf*

Display data as information

Conversion Specifier	Description
%d	Displays integer value
%f	Displays floating-point numbers
%c	Displays character

Example

```
 float result;
result = 3.123456;
printf("The value of result is %f", result);
```

Example

```
printf("\n%.4f", 3.123456);
```

Programming style

1. programming is as much of an art as it is a science
   - White space
   - Variable naming conventions
2. You should stick with a style and convention that allow you or someone else to easily read your code
   - Identify data types with a prefix
   - Use upper- and lowercase letters appropriately
   - Give variables meaningful names

scanf()

1. to receive input from users through the *scanf()* function.
2. The *scanf()* function is another built in function provided by the standard input output library *<stdio.h>;*
3. it reads standard input from the keyboard and stores it in previously declared variables
4. It takes two arguments as demonstrated next. *scanf("conversion specifier", variable);*
   - The conversion specifier argument tells *scanf()* how to convert the incoming data

Conversion Specifier	Description
%d	Receives integer value
%f	Receives floating-point numbers
%c	Receives character

Addition in C

```
# include <stdio.h>
main()
{
int iOperand1 = 0;
int iOperand2 = 0;
printf("\n\tAdder Program, by BA\n");
printf("\nEnter first operand:  ");
scanf("%d", &iOperand1);
printf("Enter second operand:  ");
scanf("%d", & iOperand2);
printf("The result is %d \n", iOperand1 + iOperand2);
}
```

Arithmetic in C

1. As demonstrated in the Adder program, C enables programmers to perform all types of arithmetic operations
2. I performed my calculation in the *printf()* function. Although this is not required, you can use additional variables and program statements to derive the same outcome

Common arithmetic operations

Operator	Description
$*$	Multiplication
$/$	Division
%	Modulus (remainder)
$+$	Addition
$-$	Subtraction

Another variation of the Adder program

```
#include <stdio.h>
main()
{
int iOperand1 = 0;
int iOperand2 = 0;
int iResult = 0;
printf("\n\tAdder Program, by BA\n");
printf("\nEnter first operand:  ");
scanf("%d", & iOperand1);
printf("Enter second operand:  ");
scanf("%d", & iOperand2); iResult = iOperand1 + iOperand2;
printf("The result is %d\n", iResult);
}
```

Note

Instead of performing the arithmetic in the *printf()* function, I've declared an additional variable called *iResult* and assigned to it the result of *iOperand1 + iOperand2* using a separate statement

Operator precedence

Order or Precedence	Description
()	Parentheses are evaluated first, from innermost to outermost
$*, /, \%$	Evaluated second, from left to right
$+$	Evaluated last

Conditions

1. Conditions (often called program control, decisions, or expressions) allow you to make decisions about program direction.
2. Learning how to use and build conditions in your program code will give you a more fluid and interactive program
   - Algorithms for conditions
   - Expressions and Conditional Operators
   - Not all programming languages, however, use the same conditional operators, so it is important to note what operators C uses

Conditional operators

Operator	Description
==	Equal (two equal signs)
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

True/False using conditional operators

Conditional operators

Expression	Result
$2 == 2$	True
$2! = 2$	False
$2 > 2$	False
$2 < 2$	False
$2 >= 2$	True
$2 <= 2$	True

Pseudo code - example

Problem statement

Turn the air conditioning on when the temperature is greater than or equal to 80 degrees or else turn it off.

Example

```
 if temperature >= 80
     Turn AC on
else
     Turn AC off
end if
```