C Programming

Boitumelo Phetla

May 23, 2019

C is a general-purpose, imperative computer programming language, supporting structured programming, lexical variable scope and recursion, while a static type system prevents many unintended operations.

1 Code Snippets

1.1 Hello World

```
/*
* Author: Boitumelo Phetla
* How to compile on Terminal
* gcc -Wall -o hello_world hello_world.c
* ./hello_world
*/
#include <stdio.h>

int main(void){
   puts("Hello world!!"); //prints out to
        console screen
   return(100);  //returns anything
   }
```

1.2 printf statement

```
/*
* Author: Boitumelo Phetla
* How to compile on Terminal
* gcc -Wall -o program program.c
* ./program
*/
#include <stdio.h>

int main(void){
   printf("Hello world!!\n"); //prints out to
        console screen
   return(0);
   }
```

1.3 scanf statement

```
/*
* Author: Boitumelo Phetla
* How to compile on Terminal
* gcc -Wall -o program program.c
* ./program
*/
#include <stdio.h>

int main(void){
   int num = 0;  //initialization

   printf("Enter a number: ");
   scanf("%d", &num);
   printf("Number: %d\n", num);
   return(0);
   }
```

1.4 Simple arithmentic Algorithm

Calculate temperature in Celsius from Farenheit inputs. $C = \frac{5}{9}(F - 32)$

```
/*
temp/
      |__________celsius.h
      |__________temp.c
*/

/*celsius.h*/

#ifndef celsius_h
#define celsius_h

//C = 5/9 * (F - 32)
float cTemp(float k){
  return (9/5 * (k - 32));
}

#endif
```

```
/*temp.c*/

#include "celsius.h"
#include <stdio.h>

int main(void){

  float k[] = {100.1, 99.9, 88.8, 77.7, 66.6,
       55.5, 44.4, 33.3, 22.2, 11.1, 5.55};

  for(int i = 0; i < sizeof(k)/sizeof(int); i++){
      printf("%.2f k = %.2f C\n", k[i],
           cTemp(k[i]));
  }

  return 0;

}



/*Output*/

100.10 k = 68.10 C
99.90 k = 67.90 C
88.80 k = 56.80 C
77.70 k = 45.70 C
66.60 k = 34.60 C
55.50 k = 23.50 C
44.40 k = 12.40 C
33.30 k = 1.30 C
22.20 k = -9.80 C
11.10 k = -20.90 C
5.55 k  = -26.45 C
```

1.5 Preprocessor

```
#include <stdio.h>
#include <math.h> //library header file
#define PI 3.1415

//A = PI^2 * r
double area(double radius);

int main(void){
  double r = 1.000003884884848884453434343;
  printf("Area(%f) = %.2f\n", r, area(r));
  return(0);
}


double area(double radius){
  return pow(PI, 2) * radius; //math
}

/*Output*/
Area(1.000004) = 9.87
```

1.6 Do-While Statement

```
#include <stdio.h>
#include <math.h> //library header file
#define PI 3.1415

//A = PI^2 * r
double area(double radius);

int main(void){

  double r = 1.000003884884848884453434343;
  do{
      printf("Area(%f) = %.2f\n", r, area(r));
           //executes nonetheless
  }while(r < 1); //terminates here condition not
       met

  return(0);
}

double area(double radius){
  return pow(PI, 2) * radius; //math
}

/*Output*/
Area(1.000004) = 9.87
```

1.7 While statement

```
#include <stdio.h>

int main(void){

  int start = 0, stop = 100, stride = 10;
  int count = 0;
  while(start <= stop){
    printf("%d\t:\t%d\n", count, start);
    count+=1;
    start+=stride;
  }
  return 0;
}

/*Output*/

0       :       0
1       :       10
2       :       20
3       :       30
4       :       40
5       :       50
6       :       60
7       :       70
8       :       80
9       :       90
10      :       100
```

1.8 Constant, While, If Statement

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

#define C 299792458     //speed of light (m/s)


float e(float m);       //e = mc^2

int main(void){

  //define sentinel as m= -1
  printf("To terminate the programe enter
       [-1]\n");

  float m = 0.0;

  printf("Enter mass [kg]: ");
  scanf("%f", &m);

  while(m > 0){
      printf("m = %.2f kg, e = %.10e m/s\n", m,
           e(m));
      printf("To terminate the programe enter
           [-1]\n");
      printf("Enter mass [kg]: ");
      scanf("%f", &m);
      if(m < 0){
        printf("Program terminated\n");
      }
  }
  return 0;
}

float e(float m){
  return (m*pow(C,2));
}


/*Output*/
To terminate the programe enter [-1]
Enter mass [kg]: 20
m = 20.00 kg, e = 1.7975103338e+18 m/s
To terminate the programe enter [-1]
Enter mass [kg]: -1
Program terminated
```

1.9 Simple getchar putchar statements

```
#include <stdio.h>
int main(){
  char c = getchar(); //input
  putchar(c);       //display
  puts("");
  return 0;
}

/*Output*/
A
A
```

1.10 Array of chars

```
#include <stdio.h>

int main(){
  int c;
  c = getchar();
  while(c != EOF){ //ctrl + D or Z
    putchar(c);
    c = getchar();
  }

  return 0;
}
```

1.11 Main function without a type

```
#include <stdio.h>

main(){
  printf("Testing\n");
  return 0;
}

/*Output*/
main.c:3:1: warning: type specifier missing,
     defaults to 'int' [-Wimplicit-int]
main(){
^
1 warning generated.
Testing
```

1.12 Static variables

```
#include <stdio.h>

/*
 * Static variables have a property of preserving
 * their value even after they are out of their
     scope.
 * static data_type variable_name =
     variable_value

   static variables

   static variables are allocated memory in data
       segment, not stack segment.

   1. data segment
   2. stack segment
   3. heap segment

   static variables are initialized as 0 in
       memory.
   static variables are used to eliminate scope
       of variables or functinos.

*/
```

```
int func();

int main(void){
   for(int i = 0; i < 5; i++){
        printf("Calling static method: %d\n",
            func());
   }
   return 0;
}

int func(){
   static int count = 0;
   count++;
   return count;
}

/*Output*/
Calling static method: 1
Calling static method: 2
Calling static method: 3
Calling static method: 4
Calling static method: 5
```

2 Control Flow

2.1 Simple If-Else statement

```
#include <stdio.h>

void num(int age);

int main(void){

  int age;
  printf("Enter your age: ");
  scanf("%d", &age);
  num(age); //calling num() function

  return 0;
}

void num(int age){
  if (age >= 18){
    printf("Welcome.\n");
  }else{
    printf("Sorry, you are under age.\n");
  }
}
```

2.2 Switch Statement

The switch statement is a multi-way decision that tests whether an expression matches one of a number of constant integer values, and branches accordingly .

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

int computeAggregate();

int main(){

  int aggregate = 0;
  aggregate = computeAggregate();
  printf("Aggregate = %d%%\n", aggregate);
  switch(aggregate){
        case 100:
        case 99:
        case 98:
        case 97:
        case 96:
        case 95:
        case 94:
        case 93:
        case 92:
        case 91:
              printf("A+\n");
              break;
        case 90:
        case 89:
        case 88:
        case 87:
        case 86:
        case 85:
        case 84:
        case 83:
        case 82:
        case 81:
               printf("A-\n");
               break;
        case 80:
        case 79:
        case 78:
        case 77:
        case 76:
        case 75:
        case 74:
        case 73:
        case 72:
        case 71:
               printf("B+\n");
               break;
        case 70:
        case 69:
        case 68:
        case 67:
        case 66:
        case 65:
        case 64:
        case 63:
        case 62:
        case 61:
               printf("B-\n");
               break;
        case 60:
        case 59:
        case 58:
        case 57:
        case 56:
        case 55:
        case 54:
        case 53:
        case 52:
```

```
        case 51:
                printf("C+\n");
                break;
        case 50:
        case 49:
        case 48:
        case 47:
        case 46:
        case 45:
        case 44:
        case 43:
        case 42:
        case 41:
                printf("C-\n");
                break;
        case 40:
        case 39:
        case 38:
        case 37:
        case 36:
        case 35:
        case 34:
        case 33:
        case 32:
        case 31:
                printf("D+\n");
                break;
        case 30:
        case 29:
        case 28:
        case 27:
        case 26:
        case 25:
        case 24:
        case 23:
        case 22:
        case 21:
                printf("D-\n");
                break;
        case 20:
        case 19:
        case 18:
        case 17:
        case 16:
        case 15:
                printf("E+\n");
                break;
        default:
                printf("E-.\n");
                break;
  }
  return 0;
}

int computeAggregate(){

    float total = 0.0;
    int count = 0;
    float grade = 0.0;
    printf("Enter a grade you obtained for a
        module: [-1 to exit]: ");
    scanf("%f", &grade);

    while(grade > 0){
        total += grade;
        count++;
        printf("Enter a grade you obtained for a
            module: [-1 to exit]: ");
        scanf("%f", &grade);
    }

    //compute average
    printf("total = %.2f, count = %d\n", total,
        count);
    printf("Aggregate = %.2f%%\n",
        (total/(count)));
    printf("Passed = %d%%\n",
        (int)(total/(count)));
    return (int)(ceil(total/(count), 2));
}


/*Output*/
Enter a grade you obtained for a module: [-1 to
    exit]: 67
Enter a grade you obtained for a module: [-1 to
    exit]: 87
Enter a grade you obtained for a module: [-1 to
    exit]: 89
Enter a grade you obtained for a module: [-1 to
    exit]: 95
Enter a grade you obtained for a module: [-1 to
    exit]: 100
Enter a grade you obtained for a module: [-1 to
    exit]: 45
Enter a grade you obtained for a module: [-1 to
    exit]: -1
total = 483.00, count = 6
Aggregate = 80.50%
B+
```

2.3 Int main Function

The main function comes in two forms:

- int main (void)
- int main (int argc, char *argv[])

2.4 Dealing with characters

1. types
2. variables
3. identifiers
4. pointers
5. arrays
6. subscripts
7. $(NULL)$

A C string is usually declared as an array of char. However, an array of char is NOT by itself a C string. A valid C string requires the presence of a terminating "null character" (a character with ASCII value 0, usually represented by the character literal " 0").

Since char is a built-in data type, no header file is required to create a C string. The C library header file <cstring> contains a number of utility functions that operate on C strings.

```
#include <stdio.h>

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

   int count = 0;
   char *string = "Hello, world!\n";

   /*print each character until we reach \0*/
   while(string[count] != '\0'){
     printf("%c", string[count++]);
   }
   return 0;
}

/*Ouput*/
Hello, world!
```

C String (or array of chars).

```
#include <stdio.h>

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

   char *str = "John Doe";

   printf("%s\n", str);

   return 0;
}

/*Output*/
John Doe
```

Constant variables cannot be mutated.

```
#include <stdio.h>
#include <math.h>
#define PI 3.1416
float area(float radius);

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

  const float r = 2.13;
  printf("Area = %.2f\n", area(r));
  //r = 1.4; <-- will give an error since r
      cannot be mutated
}

float area(float radius){
  return pow(PI, 2) * radius;
}

/*Output*/
Area = 21.02
```

Goto function.

```
#include <stdio.h>
#include <stdlib.h> //rand()
#include <string.h> //memset()
#define SIZE 1000
enum{
  VAL1='a', VAL2='b', VAL3='z'
};

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

  char a[SIZE], b[SIZE];
  int i, j;

  /* Initialise arrays so they are different
      from each other */
  memset(a, VAL1, SIZE);
  memset(b, VAL2, SIZE);

  /*Get size of array*/
  printf("size of a = %lu, b = %lu\n",
      sizeof(a)/sizeof(int),
      sizeof(b)/sizeof(int));


  /* Set a random element in each array to VALUE
      */
  a[rand()%SIZE] = VAL3;
  b[rand()%SIZE] = VAL3;
  printf("Random number: %d\n", rand()%SIZE);

  /* Search for location of common elements */
  for(i = 0; i < SIZE; ++i){
    for(j = 0; j < SIZE; ++j){
      if (a[i] == b[j]){
        goto found;
      }
    }
  }

  /* Error: match not found */
  printf("Did not find any common elements!!\n");
  return 0;

found: /*Results on success*/
      printf("a[%d] = %c, b[%d] = %c\n", i,
          a[i], j, b[j]);

}

/*Output*/
size of a = 250, b = 250
Random number: 73
a[807] = z, b[249] = z
```

Random function.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 1000

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

  for(int i = 0; i < SIZE; ++i){
    if((rand()) > 212e7){
      printf("[%d] %d, %d\n", i, rand(),
          rand()%SIZE);
    }
  }

  return 0;
}

/*Output*/
[38] 784558821, 967
[86] 1908194298, 188
[231] 462851407, 15
[240] 329863108, 249
[284] 552265483, 447
[370] 1493959603, 897
[622] 2076422667, 519
[626] 1373365825, 819
[869] 1499086275, 321
```

3 Functions and Program Stuctures

Scalable software design involves breaking a problem into sub-problems, which can each be tackled separately. Functions are the key to enabling such a division and separation of concerns. Writing programs as a collection of functions has manifold benefits, including the following.

- Functions allow a program to be split into a set of subproblems which, in turn, may be further split into smaller subproblems. This divide-and-conquer approach means that small parts of the program can be written, tested, and debugged in isolation without interfering with other parts of the program.
- Functions can wrap-up difficult algorithms in a simple and intuitive interface, hiding the implementation details, and enabling a higher-level view of the algorithm's purpose and use.
- Functions avoid code duplication. If a particular segment of code is required in several places, a function provides a tidy means for writing the code only once. This is of considerable benefit if the code segment is later altered.

Factorial function, using assert macro.

```
#include <stdio.h>
#include <assert.h>
int factorial(int a);
int main(int argc, const char *argv[]){

  int num;
  printf("Enter a number: ");
  scanf("%d", &num);
  printf("factorial(%d) = %d\n", num,
      factorial(num));
  return 0;
}

int factorial(int a){
    int result = 1;
    assert(a>=0); //check if a >= zero
    while(a){
      result *= a;
      a--;
    }

    return result;
}

/*Output*/
Enter a number: -1
Assertion failed: (a>=0), function factorial,
     file assert.c, line 15.
Abort trap: 6
```

Palindrome function

```
#include <stdio.h>
#include <string.h>

int palindrome(char *str);

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

  char word[100];

  printf("Enter a word: ");

  gets(word);

  int result = palindrome(word);

  if (result > 0){
    printf("The word is a palindrome!!\n");
  }else{
    printf("The word is not a palindrome.\n");
  }
  return 0;
}

int palindrome(char *str){

  int size = strlen(str); //string length

  for(int i = 0; i < size; i++){
    if(str[(size-1) - i] != str[i]){
        return -1; //return false
    }else{
        continue; //do not break until ’\0’
    }
  }
  return 1; //return true
}

/*Output*/

warning: this program uses gets(), which is
     unsafe.
Enter a word: racecar
The word is a palindrome!!
```