C Basics

2024年9月8日 星期日 14:07

https://wattlecourses.anu.edu.au/pluginfile.php/3749385/mod_resource/content/2/C%20Basics.pdf

C Basics

COMP2700 – BASIC C PROGRAMMING

PREPARED BY JEFFERY GUO

The following is a very quick overview of C programming language. Students are assumed to be familiar already with at least one imperative programming language.

For more details on features of C, you can consult the following sources:

- The GNU C Reference Manual: https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf

1. SOME FACTS ABOUT C PROGRAMMING LANGUAGE

- In 1988, the American National Standards Institute (ANSI) had formalized the C language.
- C was invented to write UNIX operating system.
- C is a successor of 'Basic Combined Programming Language' (BCPL) called B language.
- Linux OS, PHP, and MySQL are written in C.
- C has been written in assembly language.

2. WHY WE SHOULD USE C?

- C is the building block for many other programming languages.
- Programs written in C are highly portable.
- Several standard functions are there (like in-built) that can be used to develop programs.
- C programs are collections of C library functions, and it's also easy to add own functions to the C library.
- The modular structure makes code debugging, maintenance and testing easier.

3. C EXAMPLES

Let's go through some of the C sample codes to see how you can do programming in C.

3.1 HELLO WORLD (AS ALWAYS)

```c
#include<stdio.h>
int main()
{

   printf("Hello, World!\n");
   return 0;

}
```

The first line "#include <stdio.h>" tells the C compiler that you are going to use the standard library "stdio.h". Every C program would have a function called main. In the majority of the cases, it is the place where the program starts to execute your instructions. The function printf is a standard output syntax in C and will print out "Hello World" in this case. Since we defined **main** as a function, it will need a return value. Most C programmers choose to use 0 as the returned value for **main**.

To compile a C program, use the gcc compiler as follow:

gcc **[source file name]** -o **[output executable file name]**

If we have a helloworld.c in our current directory, we can use something like:

gcc helloworld.c –o helloworld

to run the executable, as we learnt from the previous lab. Type:

./helloworld

3.2 BASIC CONTENT TYPES & VARIABLES

```c
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
int main() {
    /*      Integer types */
    short a,b1,b2,b3; //2 bytes, -32,768 to 32,767
    a = -12345;
    printf("number stored in a is %d \n", a);
    b1 = b2 = 30000;
    b3 = b1 + b2; //overflow
    printf("number stored in b3 is %d \n", b3);
    //output a: -12345
    //b3: -5536 not 60000
    unsigned short c,d; //2 bytes, 0 to 65,535
    c = -12345;
    printf("number stored in c is %d \n", c);
    d = 56789;
    printf("number stored in d is %d \n", d);
    //c: 53191
    //d: 56789

    int e; //4 bytes, -2,147,483,648 to 2,147,483,647
    e = 2147483647;
    printf("number stored in e is %d \n", e);
    printf("Storage size for int : %d \n", sizeof(int));
    //of course, we have unsigned int
```

```
    /*
      float types
    */
    float f1,f2,f3;//4 bytes. range 1.2E-38 to 3.4E+38. precision 6 decimal places
    f1=0.123;
    f2=0.0000001;
    f3=0;
    int i;
    //f3 equals to f2 summed 10,000,000 times
    for (i=0;i<10000000;i++)
        f3=f3+f2;
    printf("number stored in f1 is %lf \n", f1);
    printf("number stored in f2 is %.7f \n", f2);
    printf("number stored in f3 is %.10f \n", f3);
    //f1: 0.123
    //f2: 0.0000001
    //f3: 1.0647674799 why this is not 1.0?

    double f4=0.0000001,f5=0;//8 bytes. range 2.3E-308 to 1.7E+308. precision 15 decimal places
    for (i=0;i<10000000;i++)
        f5+=f4;
    printf("number stored in f5 is %lf \n", f5);
    //Output is 1.000000 as expected

    /*
      char and string type
    */
    char ch='x';
    printf("character stored in ch is %c \n", ch);
    //we can define an array of char to store a sequence of characters (string)
    char str[16]="I like COMP2700";
    //although the above is 15 characters long, we need one extra for string terminator
    //OR
    char str2[]={'I',' ','l','i','k','e',' ','C','O','M','P','2','7','0','0','\0'};
    //'\0' is string terminator
    printf("string stored in str/str2 is %s \n", str2);
    //Another way of declaring a string in C (requires void function malloc in stdlib.h)
    char *str3;
    int size = 4; //one extra for '\0'
    str3 = (char*)malloc(sizeof(char)*size);
    *(str3+0) = 'a';
    *(str3+1) = 'b';
    *(str3+2) = 'c';
    *(str3+3) = '\0';
    printf("string stored in str3 is %s \n", str3);

    return 0;
}
```

Some of you may wonder what about Boolean type? In C, we don't have Boolean as basic type (C++ does). In *if* statement of C, it will treat anything equal to 0 as False and all other value as true.

3.3. ARRAYS

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(){
    //define an array syntax:
    //data_type array_name[array_size];
    int marks[5];
    srand(time(NULL));//randomize seed
    int i;
    //Initialize the array
    for (i=0;i<=4;i++)
```

```
        marks[i]=rand()%101;//integer from 0 to 100
    for (i=0;i<=4;i++){
        printf("marks[%d] = %d\n", i, marks[i]);
        }

    //define a multi dimensional array
    //data_type array_name[size1][size2]...[sizeN];
    int a[3][4] = {
        {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
        {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
        {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
        };
    //This will give us a 3 row by 4 column matrix
    //Please note that all indexes start from 0
    //If you'd like to access the array value sits at row 2 column 3.
    int value=a[1][2];
    printf("a[1][2] = %d\n", value);
    return 0;
}
```

After defining an array, it is better initialized it before using. As your array values after declaration are pretty random.

In this example, we can see there is one for loop with "{} "and one without. "{}" always works for either one or multiple command. If you choose for loop to go without {}, the compiler will assume that you are going to use the for loop with the next coming command only. This behavior is the same for *if* statement, *while* loops.

3.4. I/O

In this example, we assume there are two files, matrix1.txt and matrix2.txt, with the following contents:

matrix1.txt:

```
3 3
1 2 3
4 5 6
7 8 9
```

matrix2.txt:

```
3 3
1 1 1
2 2 2
3 3 3
```

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

int main(){
    //read two strings from stdin and compare
    char str1[256],str2[256];
    printf("Type in two strings into the standard input\nstring 1 = ");
    scanf("%s",str1);
    printf("string 2 = ");
    scanf("%s",str2);
    if (strcmp(str1,str2)==-1)
       printf("%s is smaller than %s\n",str1,str2);
       else if (strcmp(str1,str2)==0)
            printf("%s and %s are the same\n",str1,str2);
            else // ==1
```

```
            printf("%s is bigger than %s\n",str1,str2);

    //read from matrix1.txt and matrix2.txt and do the matrix multiplication in matrix3.txt
    int row1,row2,col1,col2;
    int a[10][10],b[10][10],c[10][10];
    FILE *fp1,*fp2,*fp3;
    fp1=fopen("matrix1.txt", "r");
    if(fp1 == NULL) {
       printf("File matrix1.txt not found\n");
       return 0;
    }
    fp2=fopen("matrix2.txt", "r");
    if(fp2 == NULL) {
       printf("File matrix2.txt not found\n");
       return 0;
    }

    fscanf(fp1,"%d%d",&row1,&col1);
    fscanf(fp2,"%d%d",&row2,&col2);
    int i,j,k;
    for (i=0;i<row1;i++)
        for (j=0;j<col1;j++)
            fscanf(fp1,"%d",&a[i][j]);
    for (i=0;i<row2;i++)
        for (j=0;j<col2;j++)
            fscanf(fp2,"%d",&b[i][j]);
    fclose(fp1);fclose(fp2);
    //check if you can compute matrix1 * matrix2
    if (col1 != row2){
       printf("unable to do the multiplication because of incomputable dimension.\n");
       return -1;
       }
    fp3=fopen("matrix3.txt","w");
    if(fp3 == NULL) {
       printf("Failed opening file matrix3.txt\n");
       return 0;
    }
    fprintf(fp3,"%d %d\n",row1,col2);
    for (i=0;i<row1;i++){
        for (j=0;j<col2;j++){
            c[i][j]=0;
            for (k=0;k<col1;k++)
                c[i][j]+=a[i][k]*b[k][j];
            fprintf(fp3,"%d ",c[i][j]);
            }
        fprintf(fp3,"\n");
        }
    fclose(fp3);
    return 0;
}
```

A file represents a sequence of bytes, regardless of it being a text file or a binary file. C programming language provides access on high level functions as well as low level (OS level) calls to handle file on your storage devices.

It is a good practice when you open a file for reading/writing, you will close it eventually by ***fclose()***. Otherwise, it may create deadlock or writing conflict for some cases.

3.5. FUNCTIONS

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

The C standard library provides numerous built-in functions that your program can call. For example, strcat() to concatenate two strings, memcpy() to copy one memory location to another location, and many more functions.

The return type of a function can be null. Which in C we use the keyword ***void***, this will define a null function (procedure), and you should not use ***return*** statement to return any value in this case.

```c
#include <stdio.h>
#include <stdlib.h>
void print_bigger_integer(int n1, int n2);

int sum_pass_by_value(int a,int b){
    a=a+b;
    return a;
}

int sum_pass_by_reference(int* a, int* b){
    *a=*a+*b;
    return *a;
}

int nfactorial(int n){
    if (n==1)
       return 1;
       else
           return (n * nfactorial(n-1));
}

int main(){
    int a,b;
    a=1;
    b=2;
    print_bigger_integer(a,b);
    printf("sum %d and %d is ",a,b);
    printf("%d\n",sum_pass_by_value(a,b));
printf("a = %d and b = %d\n",a,b);//a=1, b=2
    printf("sum %d and %d is ",a,b);
    printf("%d\n",sum_pass_by_reference(&a,&b));
    printf("a = %d and b = %d\n",a,b);//a=3, b=2

    a=5;
    printf("factorial of %d is %d\n",a,nfactorial(a));

    return 0;
}

void print_bigger_integer(int n1, int n2){
     printf("The large number between %d and %d is ",n1,n2);
     if (n1>n2)
        printf("%d\n",n1);
        else
            printf("%d\n",n2);
}
```

Before you are going to call a self-defined function, you must at least have the function syntax defined beforehand (Like this case: ***print_bigger_integer***). In the two sum example, the first one is passing by value, which means a and b's value has been copied and ***sum_pass_by_value*** will create two new variable called a and b to hold these two values. Any changes made to the two new variable inside ***sum_pass_by_value*** will not affect values outside that function. In our 2nd sum function, it will take a and b's memory location as input. Any operation to change the values in that memory location will inevitably change the variable value inside ***main()***.

3.6. POINTERS

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is –

type *var-name;

Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations:

```
int *ip;    /* pointer to an integer */
double *dp;   /* pointer to a double */
float *fp;   /* pointer to a float */
char *ch    /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main(){
    int* pc;
    int c;

    c=22;
    printf("Address of c:%p\n",&c);
    printf("Value of c:%d\n\n",c);
    pc=&c;
    printf("Address of pointer pc:%p\n",pc);
    printf("Content of pointer pc:%d\n\n",*pc);

    c=11;
    printf("Address of pointer pc:%p\n",pc);
    printf("Content of pointer pc:%d\n\n",*pc);

    *pc=2;
    printf("Address of c:%p\n",&c);
    printf("Value of c:%d\n\n",c);


    int i,a[5] = {1,2,3,4,5};
    int *ptr;
    ptr = a;
    for (i=0;i<5;i++)
        printf("%d",*(ptr+i));
    printf("\n");

    char *str;
    str = (char *) malloc(15);
    strcpy(str, "COMP2700");
    printf("String = %s,  Address = %p\n", str, str);
    free(str);

    return 0;
}
```

4. EXERCISES

EXERCISE 1. Read two nature numbers from standard input. Use a passing by reference function to swap the value of those two variables, and output the sum of these two numbers in base 2 to a file named binaryout.txt. For example, if a=10 and b=4 then the output would be 1110 (a string consisting of three 1's and one 0).

A template program is provided below. Complete the missing parts of the program.

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

void swap(int *a, int *b){
    // TODO: swap the numbers
}

void output(int a, int b)
{
    // TODO: output (a+b) in base 2 to a text file called binaryout.txt
}

int main(){
    int a,b;
    printf("First nature number = ");
    scanf("%d",&a);
    printf("Second nature number = ");
    scanf("%d",&b);

    if(a < 0 || b < 0) {
        printf("The input must be natural numbers\n");
        return 0;
    }

    swap(&a,&b);
    printf("After swapping, you first number is %d and second one is %d\n",a,b);
    output(a,b);
    return 0;
}
```

EXERCISE 2. Write a program in C to count the number of vowels and consonants in a string using a pointer, without using any of the functions in the string library (string.h).

```
#include <stdio.h>

int ctrV,ctrC; // global variables to store counts

void count(char *str)
{
   // TODO: update the counters ctrV and ctrC with the number of vowels
   // and consonants in str.
}

int main()
{
    char input[128];
    char *pt;

    printf("Pointer : Count the number of vowels and consonants :\n");
    printf("-------------------------------------------------------\n");
    printf(" Input a string: ");
    fgets(input, sizeof(input), stdin);
    ctrV=ctrC=0;

    count(input);

    printf(" Number of vowels : %d\n Number of consonants : %d\n",ctrV,ctrC);
    return 0;
}
```

EXERCISE 3. Try to the C code below and explain why you will not see COMP2700 gets printed out.

```c
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
int main()
{
  char* str1 = (char*)malloc(strlen("COMP2700") + 1);
  strcpy(str1, "COMP2700");
  char* str2 = str1;
  free(str1);
  printf("%s\n", str2);
  return 0;
}
```

EXERCISE 4. The code below will output "Two strings are NOT equal". Why is it the case? How do you fix it?

```c
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
int main()
{
  char* str1 = (char*)malloc(strlen("COMP2700") + 1);
  strcpy(str1, "COMP2700");
  char* str2 = (char*)malloc(strlen("COMP2700") + 1);
  strcpy(str2, "COMP2700");
  if (str1 == str2)
  {
    printf("Two strings are equal\n");
  }
  else
  {
    printf("Two strings are NOT equal\n");
  }
}
```

EXERCISE 5. Consider the following simple program that prints a hello message:

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

int main(int argc, char* argv[])
{
    char str[10];

    if(argc < 2)
    {
        printf("Usage: %s <message>\n", argv[0]);
        return 0;
    }

    strcpy(str, argv[1]);
    printf("Hello %s\n", str);

    return 0;
}
```

a. Provide an input that will crash the program. Explain why it crashes on that input.
b. How do you fix the program to avoid the crash above?

EXERCISE 6. Consider the following program that echoes an input up to a given length.

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

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

    if(argc < 3)
    {
        printf("Usage: %s <message-length> <message>\n", argv[0]);
        return 0;
    }
    length = atoi(argv[1]); // convert strings to integers
    if(length <= 0)
    {
        printf("Invalid length\n");
        return 0;
    }
    str = (char *) malloc((length+1) * sizeof(char));

    strncpy(str, argv[2], length);
    str[length] = '\0';

    printf("You said: %s\n", str);
    free(str);

    return 0;
}
```

a. Provide an input that will crash the program. Explain your answer.

b. How do you fix the program to avoid the crash above?