C-Refresher: Session 04
Terminal IO

Arif Butt

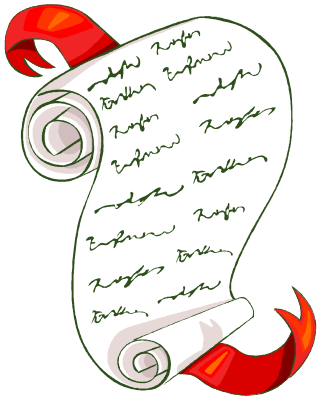
Summer 2017

I am Thankful to my student Muhammad Zubair bcsf14m029@pucit.edu.pk for preparation of these slides in accordance with my video lectures at

http://www.arifbutt.me/category/c-behind-the-curtain/

Today's Agenda

- Input Output in C
- Formatted Output using printf()
- Formatted Input using scanf()
- Unformatted I/O



Input Output(I/O) in C

- I/O in C can be done using two ways
  - High level **Library calls**
  - Low level routines(**System calls**)
- In library calls, buffer is managed implicitly, i.e. the programmer doesn't need to manage it
- In system calls, managing buffer is the programmer's responsibility

Input Output(I/O) in C(cont…)

- **Three categories of I/O**
  - Terminal I/O (will be discussed for now)
  - Disk I/O
  - Port I/O
- **Terminal I/O**

- Terminal is a visual display unit that acts as an output device for the process
- In terminal I/O, the input is taken from the keyboard and the output is displayed on the terminal
- Works in canonical and non-canonical mode

Input Output(I/O) in C(cont…)

- In **canonical mode**, input done by the user goes to the process, only when the user presses `ENTER` key. Bash shell and all C-programs normally work in canonical mode
- In **non-canonical mode**, when user presses a key it is sent to the process without the need of pressing the `ENTER` key, e.g. `vim` editor, `more` command also works in non-canonical mode

Input Output(I/O) in C(cont…)

- **Devices involved in terminal I/O**
  1. **Terminal** is used for displaying the output of the process
  2. Process takes its input from a **keyboard**
- **Two types of terminal I/O**
  1. Formatted I/O
  2. Unformatted I/O

Input Output(I/O) in C(cont…)

❑Formatted I/O

- You can specify whether to read `char`, `string`, `integer` or `float` datatype
- Functions used in formatted I/O
  - `printf()` //for output
  - `scanf()` //for input

Input Output(I/O) in C(cont…)

❑Unformatted I/O

- Input and output are treated as characters, i.e. at first, input is taken as characters, it may then be converted to required type using other functions
- Functions used in unformatted I/O
  - `getchar()` //for input
  - `gets()` //for input
  - `putchar()` //for output
  - `puts()` //for output

Formatted output using printf()

❑Prototype of printf()

```
int printf("format string", arg1,arg2,…,argn)
```

- Format string is compulsory
- Format string can be
  - Characters which are to be printed as it is on the screen
  - Escape sequence characters (/n,/t,/b,…)
  - Format specifiers for output
- Format of format specifier

```
%[some additional argument][format character]
```

Formatted output using printf()(cont…)

Format Specifier	Datatype
%d	int
%ld	long int
%lld	long long int
%f	float
%lf	double
%e	for exponential form
%c	char
%s	string
%o	octal
%x	hex
%p	printing address of pointer

Formatted output using printf()(cont…)

- `arg1,arg2,…,argn` represent arguments which can be in variable numbers
- Number of arguments must be equal to the number of format specifiers specified in the format string
  - e.g. for three format specifiers specified in the format string there must be three arguments
- Arguments must be compatible with the format specifier specified in the format string
  - e.g. `printf("%d %c %lf",char,int,double)` /*will be incorrect*/
  - `printf("%d %c %lf",int,char,double)` //will be correct

Formatted output using printf()(cont…)

```
//program using format specifier with string
#include<stdio.h>
int main(){
  printf("Hello World!\n");
  char name[40]="Muhammad Arif Butt";
  printf("%s\n",name);
  printf("%30s\n",name); //30=>field width=30
  printf("%-30s\n",name); //-=>left align the string
  printf("%.8s\n",name); //.8=>print 8 characters from string
  printf("%30.8s\n",name);
  /*30.8=>field width=30 and print 8 characters*/
  printf("%-30.8s\n",name); //-=>left align the string
  return 0;}
```

Formatted output using printf()(cont…)

- Output of the above program is:

```
Hello World!
Muhammad Arif Butt
                Muhammad Arif Butt
Muhammad Arif Butt
Muhammad
                                Muhammad
Muhammad
```

Formatted output using printf()(cont…)

```
//program using format specifier with char
#include<stdio.h>


int main(){
   char ch='A';
   printf("As Character value: %c\n",ch);
   printf("As Integer (base 10) value: %d\n",ch);
   printf("As Octal value: %o\n",ch);
   printf("As Hex value: %x\n",ch);
   printf("Address of ch: %p\n",&ch); /*may change from one time to the other*/
   return 0;
}
```

Formatted output using printf()(cont…)

- Output of the above program is:

```
As Character value: A
As Integer (base 10) value: 65
As Octal value: 101
As Hex value: 41
Address of ch: 0x7ffdd6f549b7
```

Formatted output using printf()(cont…)

```
//program using format specifier with int
#include<stdio.h>


int main(){
   int n=123456;
   printf("%d\n",n);
   printf("%10d\n",n); //field width of 10
   printf("%010d\n",n); //padded with zeroes
   printf("%-10d\n",n); //left aligned
   printf("%-010d\n",n);
   return 0;
}
```

Formatted output using printf()(cont…)

- Output of the above program is:

```
123456
     123456
0000123456
123456
123456
```

Formatted output using printf()(cont…)

```
//program using format specifier with double
#include<stdio.h>
int main(){
   double d=987.123456;
   printf("%lf\n",d);
   printf("%15lf\n",d); //field width=15
   printf("%.3lf\n",d); //print only 3 decimal places
   printf("%15.3lf\n",d); //field width=15 and decimal
places=3*/
   printf("%-15.3lf\n\n\n",d); //right aligned
```

Formatted output using printf()(cont…)

```
printf("%e\n",d); //print in exponential form
printf("%15e\n",d);
printf("%.3e\n",d);
printf("%15.3e\n",d);
printf("%-15.3e\n",d);
return 0;}
```

Formatted output using printf()(cont…)

- Output of the above program is:

```
987.123456
      987.123456
987.123
         987.123
987.123


9.871235e+02
   9.871235e+02
9.871e+02
      9.871e+02
9.871e+02
```

Formatted Input using scanf()

❑Prototype of scanf()

```
int scanf("format string",arg1,arg2,…,argn)
```

- Format string contains only format specifiers separated by spaces, as mentioned earlier for `printf()`
- No. of arguments must be same as that of the format specifiers and they should also be compatible with each other, as mentioned earlier
- `scanf()` returns the number of input items successfully matched and assigned
- **Note:** Arguments of `scanf()` are always pointers

Formatted Input using scanf()(cont…)

```
//Program showing use of scanf()
#include<stdio.h>
int main(){
  char name[50];
  printf("Enter your name: ");
  scanf("%s",name);
  int age;
  printf("Enter your age: ");
  scanf("%d",&age);
  double cgpa;
  printf("Enter your CGPA: ");
  scanf("%lf",&cgpa);
  printf("Mr. %s, you are %d years old and your CGPA is %4.2lf.
  Good Luck!\n",name,age,cgpa);
  return 0;}
```

Formatted Input using scanf()(cont…)

- An Output of the above program is:

```
Enter your name: Arif
Enter your age: 41
Enter your CGPA: 3.87
Mr. Arif, you are 41 years old and your CGPA is
3.87. Good Luck!
```

- Another output is:

```
Enter your name: Arif Butt
Enter your age: Enter your CGPA: Mr. Arif, you are
0 years old and your CGPA is 0.00. Good Luck!
```

- Here the program is displaying some garbage values for age and CGPA

Formatted Input using scanf()(cont…)

```
//another program showing use of scanf()
#include<stdio.h>
int main(){
  char name[50];
  int age;
  double cgpa;
  printf("Enter name, age and CGPA: ");
  scanf("%s %d %lf",name,&age,&cgpa); /*format specifiers to be space separated*/
  printf("Mr. %s, you are %d years old and your CGPA is %4.2lf. Good Luck!\n",name,age,cgpa);
  return 0;
}
```

Formatted Input using scanf()(cont…)

- An Output of the above program is:

```
Enter name, age and CGPA: Arif 42 3.88
Mr. Arif, you are 42 years old and your CGPA is 3.88. Good Luck!
```

- Another output is:

```
Enter name, age and CGPA: 42 3.87 Arif
Mr. 42, you are 3 years old and your CGPA is 0.87. Good Luck!
```

- Another one is:

```
Mr. Arif, you are 35261 years old and your CGPA is 490385.00. Good Luck!
```

Formatted Input using scanf()(cont…)

- **Reason for these outputs**
- `scanf()` when reading data into a string, it, by default, reads until a space or newline character `(/n)` appears in the input buffer
- When a string is entered with a space, `scanf()` reads till that space character, leaving the remaining characters in the buffer
- Then if an integer/floating point number is expected at next place from the buffer, then it creates a garbage value in the integer/floating point variable

Formatted Input using scanf()(cont…)

/*Program showing how to take a string with spaces using scanf()*/

```
#include<stdio.h>


int main(){
   char name[50];
   printf("Enter you name: ");
   scanf("%[^\n]s",name); /* \n character after ^ indicates that the scanf() should read the input till \n. Some other character can also be given in place of \n */
   printf("You are done Mr. %s\n",name);
   return 0;
}
```

Formatted Input using scanf()(cont…)

- An Output of the above program is:

```
Enter you name: Muhammad Arif Butt
You are done Mr. Muhammad Arif Butt
```

- Another output is:

```
Enter you name: Arif
You are done Mr. Arif
```

- Here, `scanf()` reads till a `\n` character is found in the input buffer

Formatted Input using scanf()(cont…)

```
//Program for showing an aspect of input buffer
#include<stdio.h>
int main(){
  int os,sp;
  printf("Enter marks in OS and SP: ");
  scanf("%d %d",&os,&sp);
  int dld;
  printf("Enter marks in DLD: ");
  scanf("%d",&dld);
  printf("Your marks is OS %d,in SP %d and in DLD are
%d\n",os,sp,dld);
  return 0;
}
```

Formatted Input using scanf()(cont…)

- An Output of the above program is:

```
Enter marks in OS and SP: 81 87
Enter marks in DLD: 90
Your marks is OS 81,in SP 87 and in DLD are 90
```

- Another output is:

```
Enter marks in OS and SP: 81 87 90
Enter marks in DLD: Your marks is OS 81,in SP 87
and in DLD are 90``
```

Formatted Input using scanf()(cont…)

- **All above outputs can be generalized by the following fact**
- `scanf()` after reading its required input leaves the remaining input(if any) entered by the user in the input buffer
- If there is some other variable, waiting next, to read the input then the user will not be prompted for new input rather the already contained input in the buffer will be read by that variable

Formatted Input using scanf()(cont…)

```
//Program for showing use of buffer by integer
#include<stdio.h>


int main(){
  int age;
  printf("Enter your age: ");
  scanf("%d",&age);
  char name[50];
  printf("Enter name: ");
  scanf("%[^\n]s",name);
  printf("Mr. %s! your age is %d\n",name,age);
  return 0;
}
```

Formatted Input using scanf()(cont…)

- An Output of the above program is:

```
Enter your age: 42
Enter name: Mr. ! your age is 42
```

- Integer type variables while reading data from the buffer leave the `\n` character in the buffer and here `name` variable reads that `\n` character from the buffer and the program does not halt for asking the name from the user
- The solution to this problem is given in the next program

Formatted Input using scanf()(cont…)

```
//Program for showing buffer in integer
#include<stdio.h>
int main(){
  int age;
  printf("Enter your age: ");
  scanf("%d",&age);
  char ch;
  scanf("%c",&ch); //will read \n character in buffer
  char name[50];
  printf("Enter name: ");
  scanf("%[^\n]s",name);
  printf("Mr. %s! your age is %d\n",name,age);
  return 0;}
```

Unformatted I/O

- There are four functions of unformatted I/O
- The two generally used functions are `getchar()` and `putchar()`
- `gets()` and `puts()` are not generally used as they cause buffer overflow problems
- In order to use `gets()` safely, you have to know exactly how many characters you will be reading, so that you can make your buffer large enough. And you will know that, only if you know exactly what data you are going to read

Unformatted I/O(cont…)

- So its unsafe to use `gets()` and `puts()` functions

```
int getchar()
```

- `getchar()` reads character from `stdin` one by one and returns it as an `unsigned character` cast to an `int`

```
int putchar(int c)
```

- `putchar()` writes `c` as an `unsigned character` on `stdout` and returns the character written as an `unsigned character` cast to an `int`

Unformatted I/O(cont…)

```
//simple program showing unformatted I/O
#include<stdio.h>
int main(){
   int ch;
   ch=getchar();
   putchar(ch);
   return 0;}
```

- The output of the program will be like
- If we press `ENTER` after typing a single character, e.g. `z`, it is printed
- If we press `ENTER` after typing a word or collection of words, e.g. `Arif Butt`, then only first character is printed i.e. `A`

Unformatted I/O(cont…)

```
#include<stdio.h>
int main(){
  int ch1,ch2;
  ch1=getchar();
  ch2=getchar();
  putchar(ch1);
  putchar(ch2);
  return 0;}
```

- In this program, if we press ENTER after typing more than one characters then the first character goes to ch1 and second to ch2
- But if we press ENTER after typing a single character, then that character goes to ch1 and \n character present in the buffer is read into ch2

Unformatted I/O(cont…)

```
#include<stdio.h>
int main(){
  int ch1,ch2;
  ch1=getchar();
  getchar(); //for reading \n character present in the buffer
  ch2=getchar();
  putchar(ch1);
  putchar(ch2);
  return 0;}
```

- Contrary to the above program, for a single input character, this program will prompt for the second character as well, as `\n` character present in the buffer will be absorbed by `getchar()` command written before taking input into `ch2`

Unformatted I/O(cont…)

/*Program reads a complete line from `stdin` and then displays it, i.e. it reads until it encounters a `\n` character at which it stops and displays the characters*/

```
#include<stdio.h>
int main(){
  int ch;
  while((ch=getchar())!='\n')
    putchar(ch);
  return 0;
}
```

Unformatted I/O(cont…)

/*Program reads until it encounters an `EOF` character(i.e. `ctrl+D`) , it actually reads until the user presses ENTER key, after that it displays the characters typed and then waits again for the next character(s) */

```
#include<stdio.h>
int main(){
   int ch;
   while((ch=getchar())!=EOF)
     putchar(ch);
   return 0;
}
```

Unformatted I/O(cont…)

/*Program shows a way of reading a string in some variable by storing it in an array for later use*/

```
#include<stdio.h>
int main(){
   char buffer[100];
   int i=0;
   while((buffer[i++]=getchar())!='\n');
   buffer[--i]='\0'; /*for replacing \n stored at this position*/
   printf("%s",buffer);
   return 0;
}
```

Unformatted I/O(cont…)

/*Program reads a string containing integers and then converts it into an integer value using `atoi()` function*/

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  char buffer[100];
  int i=0;
  while((buffer[i++]=getchar())!='\n');
  buffer[--i]='\0';
  int n=atoi(buffer); //strtol() can also be used
  printf("%d\n",++n);
  return 0;
}
```

Important Note

- All the programs written in the slides are exemplary programs for explaining the use of commands
- These were just the examples of how to use the commands, the commands may be used in various ways in different programs

SUMMARY