C Programming

Strings, Arrays,

Input, Output Statements

(a brief about input, output redirection, pipe)

Strings

- A string is a sequence of zero or more characters surrounded by double quotes
  - Eg: *“I am a string”*
  - Quotes are not part of the string
  - String constants (with only white spaces in between) are concatenated at compile time: *“hello,” “world” = “hello,world”*

Arrays

- Array is an indexed sequence of elements belonging to the same type and has a single name.
- Eg: *int total[5];* ➔ declares that total is an *int* array having 5 *int* elements.
  - *total[0]* is the first element, *total[1]* is the second element and … *total[4]* is the last element.
  - *total[index]* is an int variable where *index* gives the position of the element in the array.

Character strings

- Array of characters where last character is '\0' (null character).
- Eg: char name[10] = "cat";

  name[0] → 'c' name[1] → 'a'

  name[2] → 't' name[3] → '\0'

  name[4] to name[9] contains garbage.

character strings …

- int i;

  char name[10];

  i = 123; /* OK */

  name = “cat”; /* this assignment will not work */

/* We will learn why this is wrong */

character strings …

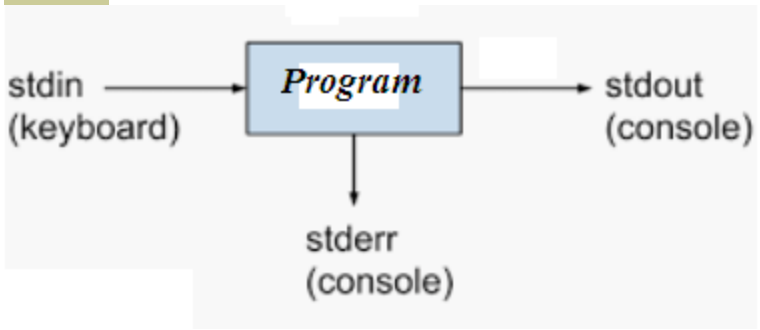
- *char name[10] = "cat"; /* initialization*/*
- *char name[10];*
  name = "cat"; / wrong assignment*/*
- assign each character as
  name[0] = 'c'; name[1] = 'a'; … name[3] = '\0';

Or one can use *strcpy(name, "cat");* a library function in *string.h* (Discussion of this is deferred)

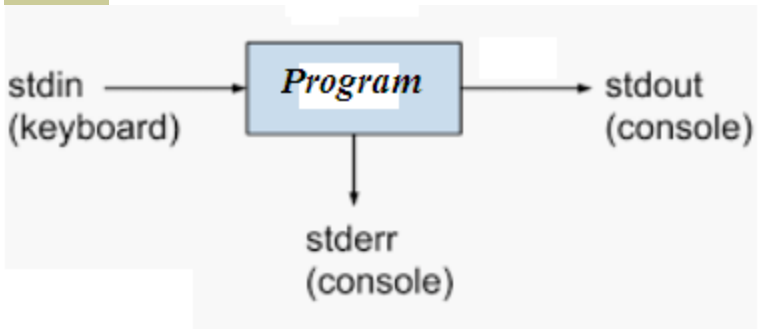
Input and Output

- Input and output facilities are not part of the C language itself !
- These are supported by a set of library functions in *stdio.h*

stdin, stdout, stderr



stdin, stdout, stderr



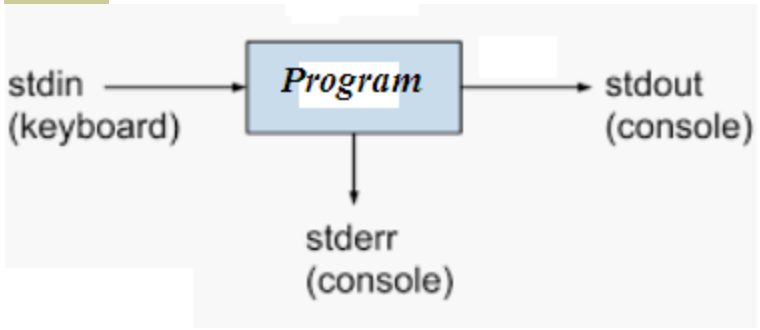
Redirecting output

Suppose now we ran the command:

```
echo ram > temp.txt
```

This should have created a new file `temp.txt`. To see the contents: `cat temp.txt`. What just happened? What does the `>` do?

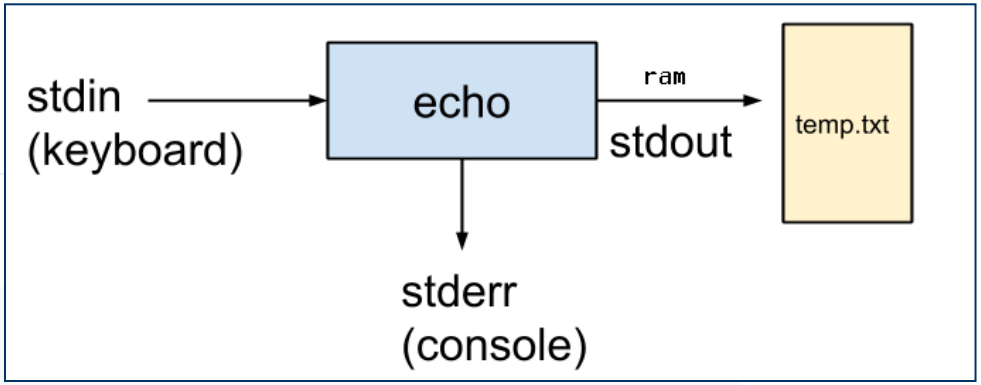
stdin, stdout, stderr



Redirecting output

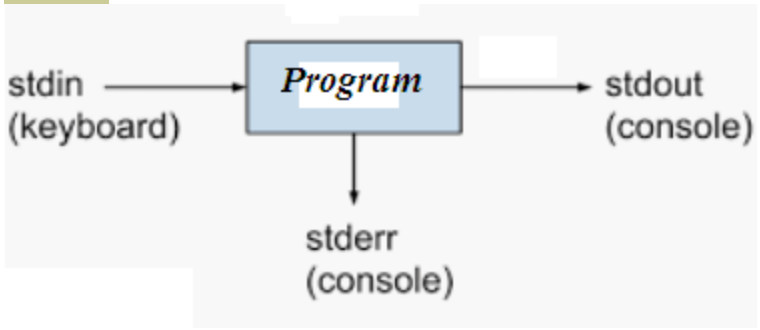
Suppose now we ran the command:

```
echo ram > temp.txt
```



This should have created a new file `temp.txt`. To see the contents: `cat temp.txt`. What just happened? What does the `>` do?

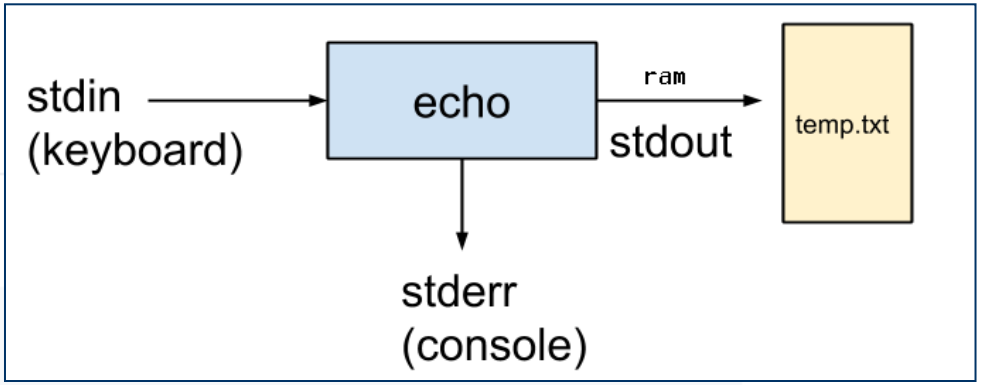
stdin, stdout, stderr



Redirecting output

Suppose now we ran the command:

```
echo ram > temp.txt
```



This should have created a new file `temp.txt`. To see the contents: `cat temp.txt`. What just happened? What does the `>` do?

```
cat temp.txt
```

This command will output what?

- To append

```
echo " bar" >> temp.txt
```

- What will be

```
cat temp.txt
```

the output?

Feeding input

Now let's run this command:

```
wc -w < temp.txt
```

You should get this output:

```
      2
```

The `wc` is a command that allows you to count things. The `-w` tells `wc` to count the number of words. The `<` told `wc` to feed its input from `temp.txt`, essentially making `temp.txt` the `stdin` of wc:

Feeding input

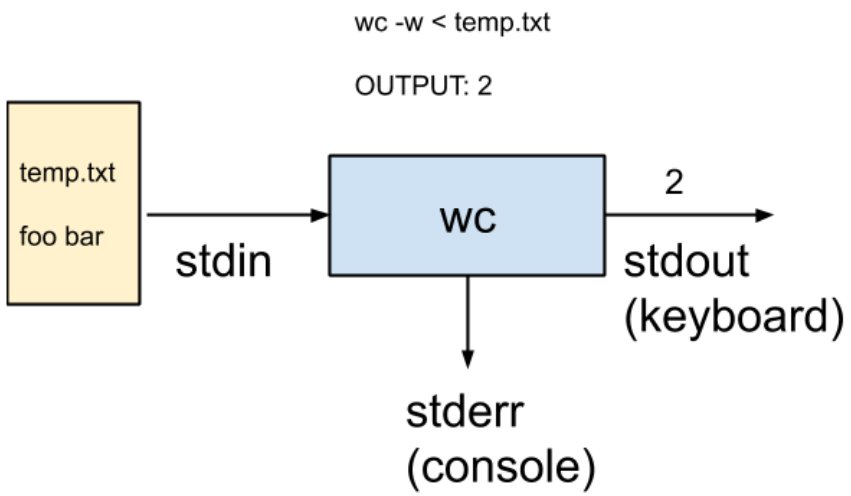
Now let's run this command:

```
wc -w < temp.txt
```

You should get this output:

```
       2
```

The `wc` is a command that allows you to count things. The `-w` tells `wc` to count the number of words. The `<` told `wc` to feed its input from `temp.txt`, essentially making `temp.txt` the `stdin` of wc:

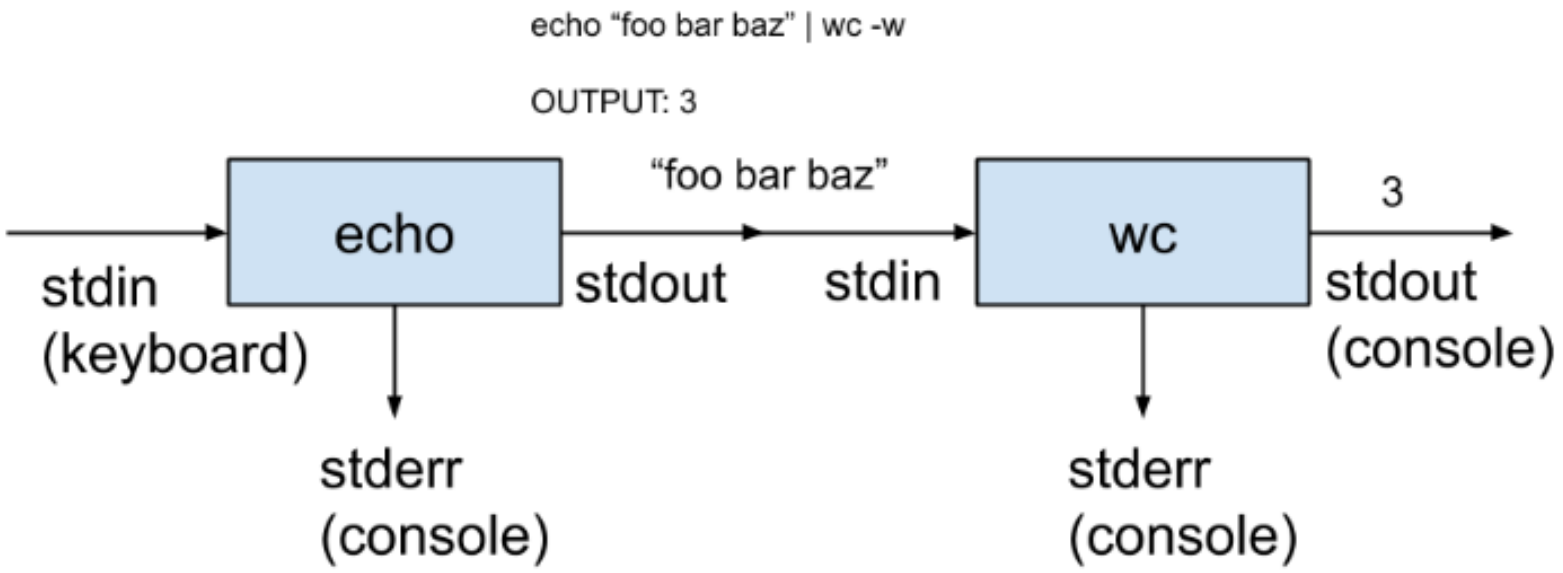


Pipes

A natural question to ask now is: Can we feed the output of one program as input into another? And the answer is yes! We do this using something called a **pipe**. Run this command:

```
echo "foo bar baz" | wc -w
```

You should get `3` as your output. The `|` character tells your terminal to feed the stdout of the `echo` command as stdin into the `wc` command:



Using pipes, we can chain together many small simple programs together to do very powerful things (like our scraper example)!

getchar( ) and putchar( )

```
char c;              /* declaration */
c = getchar( );      /* reads a character from
                        stdin (keyboard)*/
putchar(c);          /* writes the character stored
                        in the variable c on the
                        stdout (screen)*/
```

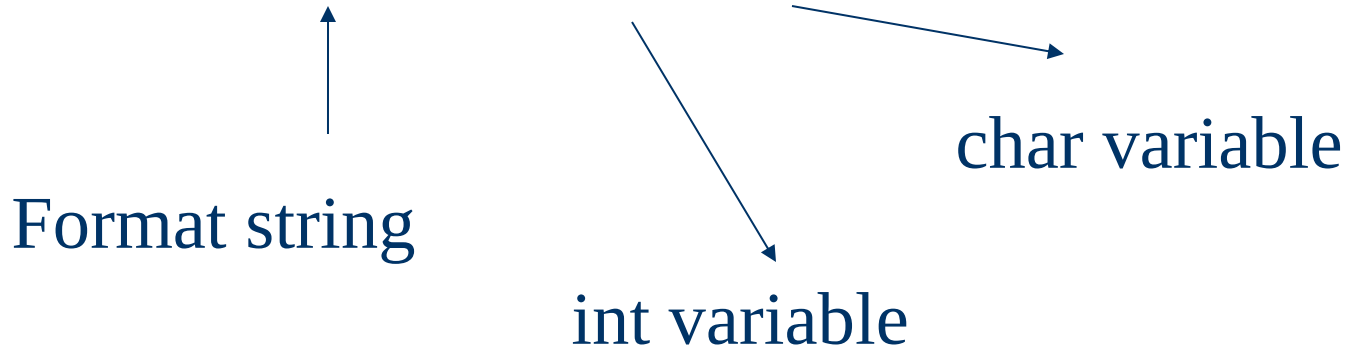
scanf (formatted input)

```
int i;
char ch;
scanf("%d", &i); /* read an integer from
                    stdin into i */
scanf("%c", &ch); /* read a char from stdin
                     into ch */
```

scanf (formatted input)

/* both can be read in a single scanf */

scanf("%d %c", &i, &ch);



Why there is '*&*' character before *i* and *ch* ?

Why ‘&’ in scanf ?

- Every variable should have a location in the memory.
- If *i* is variable’s name then *&i* is its address in the memory.
- *scanf* should be given the *addresses* of the locations where it should store the read values.

scanf

- Different values (that is given as input) should be separated with a space or newline.
  - More about this is deferred.

printf (formatted output)

```
int i; char ch;
i = 125;
ch = ‘a’;
printf(“%d  %c \n”, i, ch);
printf(“%c\n%d”,ch,i);
```

```
$./a.out
125 a
a
125
$
```

Output on the screen

printf

```
char ch = ‘a’;
printf(“%d”, ch);
```

What will be the output?

How to read or write a float?

%f → float

```
float num;
scanf("%f ", &num);
printf("%f ", num);
```

How to read or write a string?

```
char str[64];
printf("what is your name:");
scanf("%s", str);
printf("Hello … %s \n", str);
```

Output on screen

```
$./a.out
What is your name: Ram
Hello … Ram
$
```

Formatted Output with printf()

Format Conversion Specifiers:

d -- displays a decimal (base 10) integer

l -- used with other specifiers to indicate a "long"

e -- displays a floating point value in exponential notation

f -- displays a floating point value

c -- displays a single character

s -- displays a string of characters

Strings

- Why we have not used ‘&’ while reading a string ?

  Findout the answer.

More about scanf, printf, etc later ……

Input Function getc( )

- getc(*file) ;

- This function is similar to getchar( ) except the input can be from the keyboard or a file.
- Example:

```
char ch;
ch = getc (stdin);        /* input from keyboard */
ch = getc (fileptr);      /* input from a file */
```

Output Function putc()

- putc (char, *file) ;

- This function is similar to putchar ( ) except the output can be to the screen or a file.
- Example:

```
char ch;
ch = getc (stdin);          /* input from keyboard */
putc (ch, stdout);          /* output to the screen */
putc (ch, outfileptr);      /*output to a file */
```