

# A Taste of C

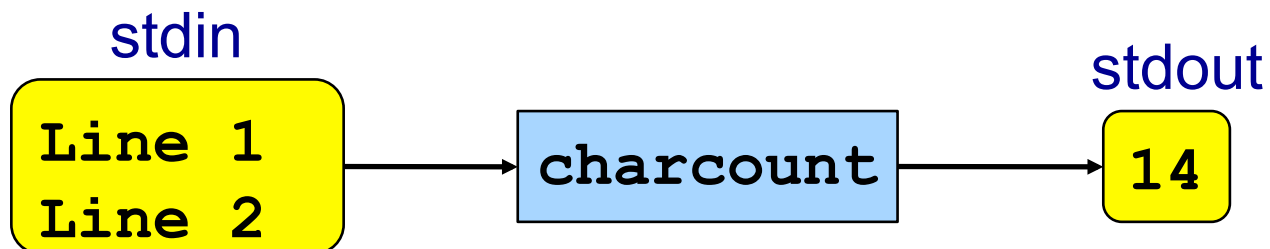




# The “charcount” Program

## Functionality:

- Read all chars from stdin (standard input stream)
- Write to stdout (standard output stream) the number of chars read





# The “charcount” Program

The program:

charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{   int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {   charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

# “charcount” Building and Running



```
$ gcc217 charcount.c -o charcount
$ charcount
Line 1
Line 2
^D
14
$
```

What is this?  
What is the effect?

# “charcount” Building and Running



```
$ cat somefile  
Line 1  
Line 2  
$ charcount < somefile  
14  
$
```

What is this?  
What is the effect?

# “charcount” Building and Running



```
$ charcount > someotherfile  
Line 1  
Line 2  
^D  
$ cat someotherfile  
14
```

What is this?  
What is the effect?



# “charcount” Building and Running in Detail

## Question:

- Exactly what happens when you issue the command  
`gcc217 charcount.c -o charcount`

## Answer: Four steps

- Preprocess
- Compile
- Assemble
- Link



# “charcount” Building and Running in Detail

## The starting point

### charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{   int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {   charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

- C language
- Missing definitions of getchar() and printf()





# Preprocessing “charcount”

Command to preprocess:

- `gcc217 -E charcount.c > charcount.i`

Preprocessor functionality

- Removes comments
- Handles **preprocessor directives**



# Preprocessing “charcount”

## charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{   int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {   charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

Preprocessor replaces  
#include <stdio.h>  
with contents of  
/usr/include/stdio.h



# Preprocessing “charcount”

## charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{   int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {   charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

Preprocessor  
removes comment



# Preprocessing “charcount”

## The result

charcount.i

```
...  
int getchar();  
int printf(char *fmt, ...);  
...  
int main(void)  
{  int c;  
  int charCount = 0;  
  c = getchar();  
  while (c != EOF)  
  {  charCount++;  
    c = getchar();  
  }  
  printf("%d\n", charCount);  
  return 0;  
}
```

Why `int` instead  
of `char`?

- C language
- Missing comments
- Missing preprocessor directives
- Contains code from `stdio.h`
  - **Declarations** of `getchar()` and `printf()`
- Missing **definitions** of `getchar()` and `printf()`



# Compiling “charcount”

## Command to compile:

- `gcc217 -S charcount.i`

## Compiler functionality

- Translate from C to assembly language
- Use function declarations to check calls of `getchar()` and `printf()`



# Compiling “charcount”

## charcount.i

```
...  
int getchar();  
int printf(char *fmt, ...);  
...  
int main(void)  
{   int c;  
    int charCount = 0;  
    c = getchar();  
    while (c != EOF)  
    {   charCount++;  
        c = getchar();  
    }  
    printf("%d\n", charCount);  
    return 0;  
}
```

- Compiler sees function declarations
- So compiler has enough information to check subsequent calls of `getchar()` and `printf()`



# Compiling “charcount”

## charcount.i

```
...  
int getchar();  
int printf(char *fmt, ...);  
...  
int main(void)  
{  int c;  
    int charCount = 0;  
    c = getchar();  
    while (c != EOF)  
    {  charCount++;  
        c = getchar();  
    }  
    printf("%d\n", charCount);  
    return 0;  
}
```

- Definition of main() function
- Compiler checks calls of getchar() and printf() when encountered
- Compiler translates to assembly language



# Compiling “charcount”

The result: charcount.s

```
.section ".rodata"
format:
.string "%d\n"
.section ".text"
.globl main
.type main,@function
main:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
    call getchar
loop:
    cmpl $-1, %eax
    je endloop
    incl -4(%ebp)
    call getchar
    jmp loop
endloop:
    pushl -4(%ebp)
    pushl $format
    call printf
    addl $8, %esp
    movl $0, %eax
    movl %ebp, %esp
    popl %ebp
    ret
```

- Assembly language
- Missing definitions of getchar() and printf()





# Assembling “charcount”

Command to assemble:

- `gcc217 -c charcount.s`

Assembler functionality

- Translate from assembly language to machine language



# Assembling “charcount”

The result:

charcount.o

Machine language  
version of the  
program

No longer human  
readable

- Machine language
- Missing definitions of  
getchar() and printf()



# Linking “charcount”

## Command to link:

- `gcc217 charcount.o -o charcount`

## Linker functionality

- Resolve references
- Fetch machine language code from the standard C library (/usr/lib/libc.a) to make the program complete



# Linking “charcount”

The result:

charcount

Machine language  
version of the  
program

No longer human  
readable

- Machine language
- Contains definitions of  
getchar() and printf()

Complete! Executable!



# Running “charcount”

Command to run:

- `charcount < somefile`



# Running “charcount”

Run-time trace, referencing the original C code...

## charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{   int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {   charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

Computer allocates space  
for c and charCount in the  
stack section of memory

Why int instead  
of char?



# Running “charcount”

Run-time trace, referencing the original C code...

## charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{   int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {   charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

- Computer calls `getchar()`
- `getchar()` tries to read char from stdin
  - Success => returns char (within an int)
  - Failure => returns **EOF**

**EOF** is a special non-char value that `getchar()` returns to indicate failure



# Running “charcount”

Run-time trace, referencing the original C code...

## charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{   int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {   charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

Assuming  $c \neq \text{EOF}$ ,  
computer increments  
charCount





# Running “charcount”

Run-time trace, referencing the original C code...

## charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{   int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {   charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

Computer calls `getchar()`  
again, and repeats



# Running “charcount”

Run-time trace, referencing the original C code...

## charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{   int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {   charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

- Eventually getchar() returns EOF
- Computer breaks out of loop
- Computer calls printf() to write charCount



# Running “charcount”

Run-time trace, referencing the original C code...

## charcount.c

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void)
{   int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF)
    {   charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```

- Computer executes return stmt
- Return from main() terminates program

Normal execution => return 0 or **EXIT\_SUCCESS**

Abnormal execution => return **EXIT\_FAILURE**



# Other Ways to “charcount”

1 `for (c=getchar(); c!=EOF; c=getchar())  
 charCount++;`

2 `while ((c=getchar()) != EOF)  
 charCount++;`

Which way  
is best?

3 `for (;;)
{
 c = getchar();
 if (c == EOF)
 break;
 charCount++;
}`

4 `c = getchar();
while (c!=EOF)
{
 charCount++;
 c = getchar();
}`



# Review of Example 1

## Input/Output

- Including `stdio.h`
- Functions `getchar()` and `printf()`
- Representation of a character as an integer
- Predefined constant `EOF`

## Program control flow

- The `for` and `while` statements
- The `break` statement
- The `return` statement

## Operators

- Assignment: `=`
- Increment: `++`
- Relational: `==` `!=`