4010-350 Personal SE

Introduction to C

A Bit of History

- Developed in the early to mid 70s
  - Dennis Ritchie as a systems programming language.
  - Adopted by Ken Thompson to write Unix on a the PDP-11.
- At the time:
  - Many programs written in assembly language.
  - Most systems programs (compilers, etc.) in assembly language.
  - Essentially ALL operating systems in assembly language.
- Proof of Concept
  - Even small computers could have an OS in a HLL.
  - Small: 64K bytes, 1μs clock, 2 MByte disk.
  - We ran *5 simultaneous users* on this base!

But Efficiency Wasn't Cheap in the 70s

- Compiler development still art as much as science.
- Code optimization in its infancy.
- C as a consquence:
  - Has types (but they can be easily ignored).
  - Has no notion of objects (just arrays and structs).
  - Permits pointers to arbitrary locations in memory (Scout's Honor Programming).
  - Has no garbage collection – it's the programmer's job to manage memory.
- That is, C is the band saw of programming languages:
  - Very powerful and doesn't get in your way.
  - Very dangerous and you can cut off your fingers.

What Java Borrowed From C

- { and } for grouping.
- Prefix type declaration (e.g., int i vs. i : int).
- Control structures (mostly)
  - if, switch
  - while, for
- Arithmetic (numeric) operations:
  - ++ and -- (prefix and suffix)
  - *op*= (e.g. += *=, etc.)
  - + - * / %
- Relational & boolean operators:
  - < > <= >= != ==
  - ! || &&

Things Uniquely C

- Today
  - No classes – just functions & data.
  - Characters are just small integers.
  - No booleans.
  - Limited visibility control via #include and separate compilation.
  - Simple manifest constants via #define
- Later
  - Array size fixed at compile time.
  - Strings are just constant arrays.
  - Simple data aggregation via structures (**struct**)
  - And, last but not least – POINTERS!!!

Functions & Data

- C functions – like methods free from their class.
- The most important function: main
- Example: Hello, world

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

int main( ) {
    puts( "Hello, world!" ) ;
    return 0 ;
}
```

Functions & Data

- C functions – like methods free from their class.
- The most important function: main
- Example: Hello, world

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

int main( ) {
    puts( "Hello, world!" ) ;
    return 0 ;
}
```

Includes interface information to other modules

Similar to import in Java

But done textually!!

Functions & Data

- C functions – like methods free from their class.
- The most important function: main
- Example: Hello, world

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

int main( ) {
    puts( "Hello, world!" ) ;
    return 0 ;
}
```

stdlib
atoi, atol, atof
memory allocation
abort, exit, system, atexit
qsort, bsearch [advanced]

Functions & Data

- C functions – like methods free from their class.
- The most important function: main
- Example: Hello, world

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

int main( ) {
    puts( "Hello, world!" ) ;
    return 0 ;
}
```

stdio

getchar, fgetc, putchar, fputc
printf, fprintf, sprintf
gets, puts, fgets, fputs
scanf, fscanf, sscanf

Functions & Data

- C functions – like methods free from their class.
- The most important function: main
- Example: Hello, world

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

int main( ) {
    puts( "Hello, world!" ) ;
    return 0 ;
}
```

Every C program has a **main** function – the first function called.

main returns exit status.
    0 = ok
    anything else = abnormal.

Functions & Data

- C functions – like methods free from their class.
- The most important function: main
- Example: Hello, world

puts, from **stdio**, prints a string and appends a newline ('\n').

Strings are simpler in C than Java.

C strings are just constant arrays.

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

int main( ) {
   puts( "Hello, world!" ) ;
   return 0 ;
}
```

Characters are Small Integers

- Consider the following C constants"

  `'a'        97        0141        0x61`

- In C they are all the *<u>same value</u>* – a small positive **int**.
- That is, character constants are just small integers.
  - Use the notation that expresses what you are doing:
  - If working with numbers, use 97 (or 0141 / 0x61 if bit twiddling).
  - If working with letters, use 'a'.
  - Question: what is 'a' + 3?
  - Question: if ch holds a lower case letter, what is ch - 'a'?
- Escape sequences with backslash:
  - `'\n'` == newline, `'\t'` == tab, `'\r'` == carriage return
  - `'\`***ddd***`'` == character with octal code ***ddd*** (the ***d***'s are digits 0-7).
  - `'\0'` == NUL character (end of string in C).

Integer Types in C

char	one byte = 8 bits - possibly signed
unsigned char	one byte unsigned
short	two bytes = 16 bits signed
unsigned short	two bytes unsigned
int	"natural" sized integer, signed
unsigned int = unsigned	"natural" sized integer, unsigned
long	four bytes = 32 bits, signed
unsigned long	four bytes, unsigned
long long	eight bytes = 64 bits, signed
unsigned long long	eight bytes, unsigned

Another Example – Count Punctuation

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

int main( ) {
    int tot_punct = 0 ; // declare & init. a local variable
    int nchar ;         // next character read

    while( (nchar = getchar()) != EOF ) {
        if( ispunct(nchar) ) {
            ++tot_punct ;
        }
    }

    printf( "%d punctuation characters\n", tot_punct ) ;
    return 0 ;
}
```

Another Example – Count Punctuation

ctype

isalnum, isalpha, isdigit, iscntrl
islower, isupper, ispunct, isspace
isxdigit, isprint
toupper, tolower

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

int main( ) {
    int tot_punct = 0 ; // declare & init. a local variable
    int nchar ;         // next character read

    while( (nchar = getchar()) != EOF ) {
        if( ispunct(nchar) ) {
            ++tot_punct ;
        }
    }

    printf( "%d punctuation characters\n", tot_punct ) ;
    return 0 ;
}
```

Another Example – Count Punctuation

Next character from standard in.
Why **int** and not **char**?
Because EOF is negative!

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

int main( ) {
    int tot_punct = 0 ; // declare & init. a local variable
    int nchar ;         // next character read

    while( (nchar = getchar()) != EOF ) {
        if( ispunct(nchar) ) {
            ++tot_punct ;
        }
    }

    printf( "%d punctuation characters\n", tot_punct ) ;
    return 0 ;
}
```

Another Example – Count Punctuation

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

int main( ) {
    int tot_punct = 0 ; // declare & init. a local variable
    int nchar ;         // next character read

    while( (nchar = getchar()) != EOF ) {
        if( ispunct(nchar) ) {
            ++tot_punct ;
        }
    }

    printf( "%d punctuation characters\n", tot_punct ) ;
    return 0 ;
}
```

Common C idiom:
  Get & assign value
  Compare to control flow
= vs. == can kill you here.

Another Example – Count Punctuation

EOF defined in **stdio.h** as (-1)
Not a legal character.
Signals end-of-file on read.

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

int main( ) {
    int tot_punct = 0 ; // declare & i   . a local variable
    int nchar ;         // next charac er read

    while( (nchar = getchar()) != EOF ) {
        if( ispunct(nchar) ) {
            ++tot_punct ;
        }
    }

    printf( "%d punctuation characters\n", tot_punct ) ;
    return 0 ;
}
```

Another Example – Count Punctuation

Helper function from **ctype**
True iff nchar is punctuation.

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

int main( ) {
    int tot_punct = 0 ; // declare & init. a local variable
    int nchar ;         // next character read

    while( (nchar = getchar()) != EOF ) {
        if( ispunct(nchar) ) {
            ++tot_punct ;
        }
    }

    printf( "%d punctuation characters\n", tot_punct ) ;
    return 0 ;
}
```

Another Example – Count Punctuation

Formatted output to standard out.

printf = **print f**ormatted

1st argument is format string

Remaining arguments are printed according to the format.

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

int main( ) {
    int tot_punct = 0 ; // de        & init. a local variable
    int nchar ;         // n       haracter read

    while( (nchar = getch      ) != EOF ) {
        if( ispunct(ncha     ) {
            ++tot_punct ;
        }
    }

    printf( "%d punctuation characters\n", tot_punct ) ;
    return 0 ;
}
```

Short Digression on Printf

- Format string printed as is except when encounters '%'

– %d	print integer as decimal
– %f	print floating point (fixed point notation)
– %e	print floating point (exponential notation)
– %s	print a string
– %c	print integer as a character
– %o / %x	print integer as octal / hexadecimal

- Format modifiers - examples

– %***n.m***f	at least ***n*** character field with ***m*** fractional digits
– %***n***d	at least ***n*** character field for a decimal value.

- Example:

```
printf("%d loans at %5.2f%% interest\n",nloans, pct) ;
```

- See the stdio.h documentation for more on format control.

Boolean = Integer

- There is no boolean type in C.
- 0 is **false**, *everything* else is **true**.

– False:	0	0.0	'\0'	NULL (0 pointer).
– True:	1	'a'	3.14159	

- The result of a comparison operator is 0 or 1.
- Many programmers define symbolic constants:

```
#define TRUE  (1)
#define FALSE (0)
```

- Pet Peeve:

BAD

```
if ( value < limit ) {
   return TRUE ;
} else {
    return FALSE ;
}
```

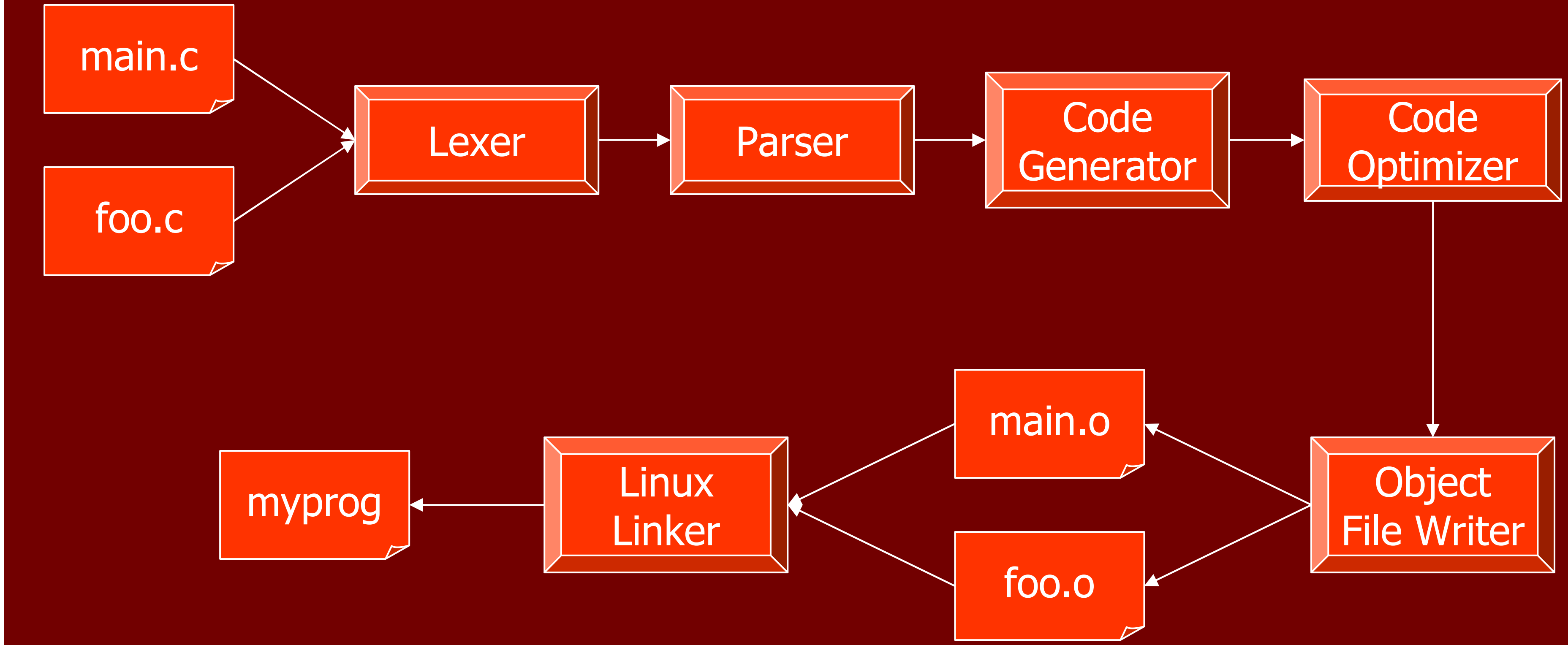
GOOD

```
return value < limit ;
```

Compilation

- Our systems use the GNU C compiler (gcc)
- The compilation process with two files (main.c, foo.c)

```
gcc –o myprog main.c foo.c
```



Compilation

- Problems can occur all along the line:
  - Unterminated comments can throw off the lexer.
  - Syntax errors are detected by the parser.
  - The code generator / optimizer can generate bad code (highly unlikely).
  - The linker may not be able to resolve all the external references.
- Notes on linking:
  - Every object file has a table of contents.
  - Some of the names are defined in the file (e.g., main).
  - Some are needed from another file (e.g., printf).
  - The linker tries to resolve these BUT:
    - It may not be able to find a symbol it needs (missing file?)
    - It may find two definitions of a symbol (name conflict).