ECE 220: Computer Systems & Programming

Lecture 5: Introduction to C

Ujjal Kumar Bhowmik

- MP2 due this Thursday.
- Mock Quiz1: 1/30 – 2/1
- Quiz1: 2/5 – 2/7

Introduction to C

Overview

C is a general-purpose high-level computer programming language

- Provides an abstraction from the underlying hardware
- Is independent of ISA
- Is *expressive*, meaning that complex tasks can be expressed with a small amount of code
- Is much more readable than assembly code
  - symbolic names are used instead of memory locations and registers to refer to values
  - operators are used to manipulate values
    - but note that some operators are taken directly from the assembly language, e.g. ++

Overview (cont.)

C is a procedural language

- the programmer specifies an explicit sequence of steps to follow to produce a result
- the program is composed of procedures, also called a function, or routine, or subroutine

C programs are compiled rather that interpreted

- Compiler translates a program written in C into machine code that is directly executable by the processor for which it is compiled
- For comparison, interpreted programs are executed by another program, called an interpreter. They are not translated into binary instructions

C language was invented in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system

- Was standardized in 1988, the standard is called ANSI C (for American National Standards Institute)
- In 1990, the ANSI C standard with some minor modifications was adopted by the International Organization for Standardization. This version is called C90
- In this course we will study **ANSI C**

Basic C program structure

```
/* Compute area of a circle
   INPUT: radius; OUTPUT: area, printed to the terminal */

#include <stdio.h>
#define PI 3.141576f
int main()
{
    float r;       /* radius */
    float A;   /* area and perimeter */

    printf("Enter radius: ");
    scanf("%f", &r);

    A = PI * r * r;      /* area */

    printf("A=%f \n", A);
    return 0;  /* terminate program return 0 to the operating system*/
}
```

- **Comments**

```
/* this is a comment. Not to be compiled. Ends with */
```

- **pre-processor directives begin with #**
  - `#include <stdio.h>`
    - Instructs the pre-processor to copy content of stdio.h (header file) into the source code
    - stdio.h header file includes function declarations necessary to use standard I/O functions in C
    - almost all programs will need to include this header file
    - other examples of include files are math.h, stdlib.h, etc.
    - <stdio.h> and other header files included in <> are located in some well-defined place in the file system known to the compiler
    - Header files located in the current directory or the directory provided to the compiler by the user are enclosed in “”, e.g., “mydefs.h”
  - **`#define PI 3.1416f`**

Directs the pre-processor to replace all instances of string PI in the file being pre-processed with the value of 3.1416f

Compiling C Program

Preprocessor

- macro substitution
- conditional compilation
- "source-level" transformations
  - output is still C

Compiler

- generates object file
  - machine instructions

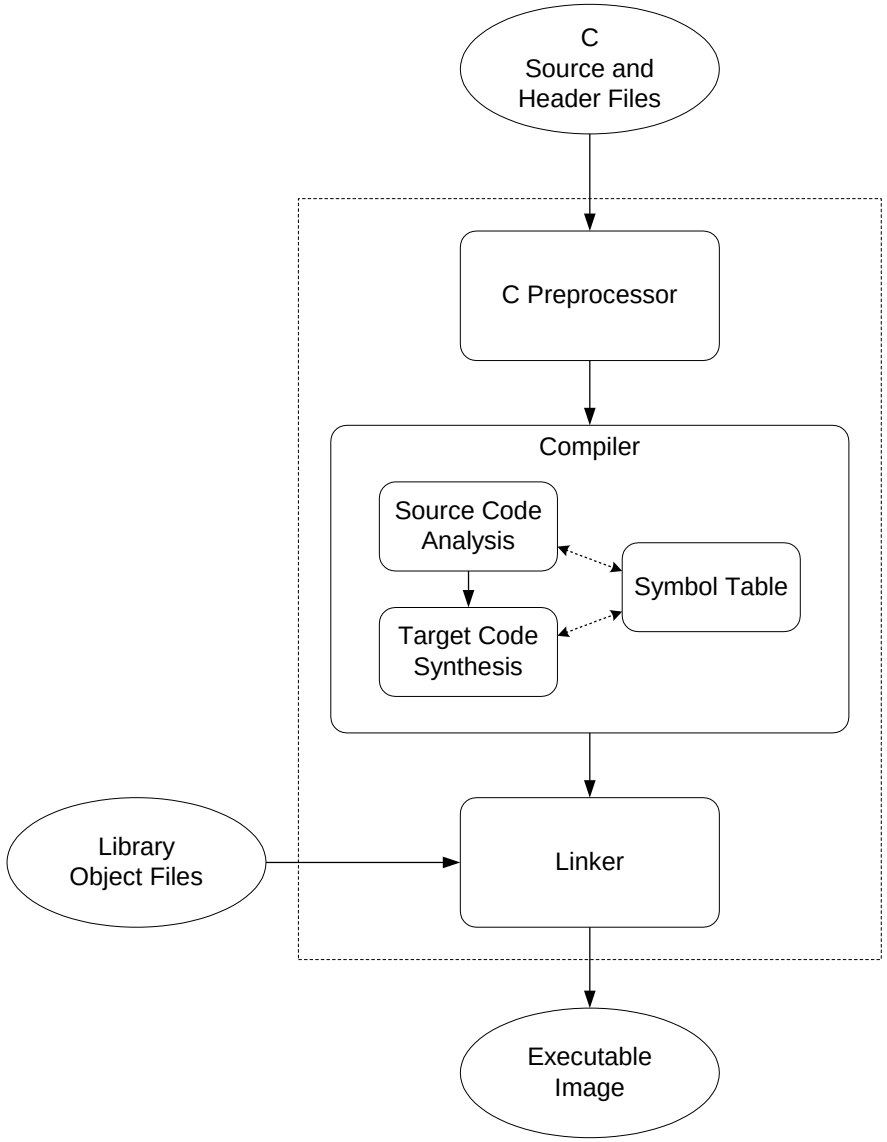
Linker

- combine object files

(including libraries)

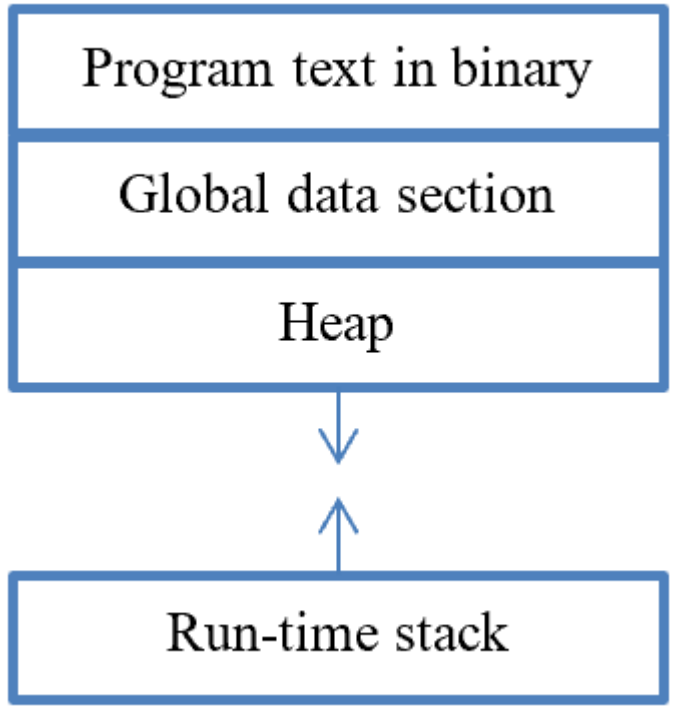
into executable image

✓ **gcc compiler – invoke all these tools**



Variables in C

- **int** (long, long long, unsigned), can also use hex representation 0xD
- **float** (double)
- **char**
- ***const*** - constant qualifier
- ***static*** - static qualifier
- **Storage class**: static vs. automatic
- **Scope**: local vs. global



Example: Global Variable

```c
#include<stdio.h>
int Global = 5;

int main()
{
    int Local = 1;    /* local to main */
    printf("Global %d Local %d\n", Global, Local);
    {
        int Local = 2;    /* local to this block */
           Global = 4;        /* change global variable */
        printf("Global %d Local %d\n", Global, Local);
    }
    printf("Global %d Local %d\n", Global, Local);

    return 0;
}
```

Output:

```
Global 5 Local 1
Global 4 Local 2
Global 4 Local 1
```

Operators

- Expression vs. Statement
- The Assignment Operator (=):
- ‘=‘ vs. ‘==‘
- Arithmetic Operators: *, +, -, /, % (modulus)
- Order of evaluation:
- precedence `x = 2+3*4;`
- associativity `x = 2+3-4+5;`
- parentheses `x = a*(b + c)*d/2;`
- Logical Operators:
- Bitwise Operators:
- Relational Operators:

Operators (cont.)

Increment/Decrement Operators: ++, -- (pre vs. post)

```
example 1: x = 4; y = ++x;
example 2: x = 4; y = x++;
```

What is the value of x and y after increment?

```
example 1
example 2
```

- Special operator (conditional):

Variable = condition ? value_if_true : value_if_false;

```
example: M = (x<y) ? 3 : 5
/* if x<y, M =   ; otherwise, M=    */
```

```
Expression with multiple operators
```

- Example: y = x & z + 3 || 9 – w % 6;

/* y = (x & (z + 3)) || (9 – (w%6)); */

Compound Assignment Operators:

- `a += b; a = a + b;`

Example

```
#include <stdio.h>
int main(){
/* declare integer variables a, b and c */

/* set a to 2, set b to a^3*/

/* left shift b by a number of bits */

/* perform bitwise AND on a and b, store the
result to c */

/* print c */

return 0;
}
```

Example

```
/* Compute perimeter of a circle; Given PI=3.141576
   INPUT: radius; OUTPUT: area, printed to the terminal */

//preprocessor directives

int main()
{
    // declare variable

    // prompt user to give input
    // get input

/* calculate perimeter */

    //print result

    return 0;  /* terminate program return 0 to the operating system*/
}
```