High Performance Computing
Lecture 7

Matthew Jacob

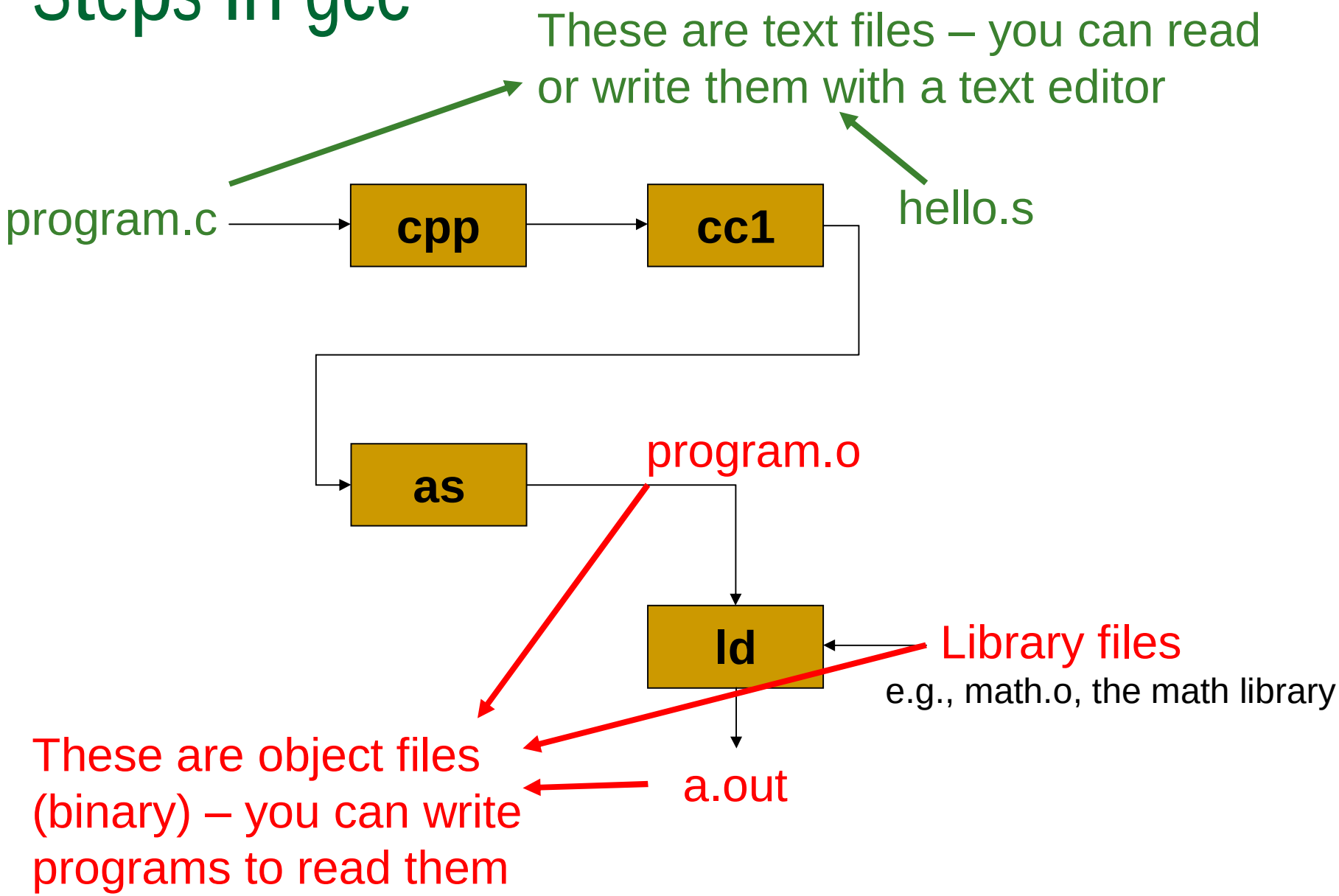
Indian Institute of Science

Recall: C Program to a.out

% gcc program.c

- program.c: File containing program written in the C programming language
- a.out: File containing executable equivalent program in machine language

Steps in gcc


These are text files – you can read or write them with a text editor
program.c
cpp
cc1
hello.s
as
program.o
ld
Library files
e.g., math.o, the math library
These are object files (binary) – you can write programs to read them
a.out

Steps in gcc

- cpp: C pre-processor
  - Pre-processing of #include, #define, …
  - Output: an expanded C program
- cc1: C compiler
  - Output: an equivalent assembly language program
  - Almost like machine language but readable
- as: Assembler
  - Output: an equivalent machine language program
- ld: Linkage editor

Sample program.c

```
#include<stdio.h>
#include<math.h>
float a[100];
main() {
    int i;
    float sum;
    for(i=0, sum=0.0; i<100; i++) {
        a[i] = sqrt(a[i]);
        sum += a[i];
    }
    printf("sum = %4.2f\n", sum);
}
```

Corresponding program.s

```
 .section .bss, 8, 0x00000003, 0, 8
.bss:

        .section .lit8, 1, 0x30000002, 8, 8
.lit8:

        .section .rodata, 1, 0x00000002, 0, 8
.rodata:

        .section .bss
        .origin 0x0
        .align  0
        .globl  a
        .type    a, stt_object
        .size    a, 400
```

Assembler directives

Assembly Representation.

```
a:      # 0x0
        .dynsym a       sto_default
        .space  400
        .section .text

 # Program Unit: main
        .ent    main
        .globl  main
main:   # 0x0
        .dynsym main    sto_default
        .frame  $sp, 16, $31
        .mask   0x80000000, -8
        # gra_spill_temp_0 = 0
        # gra_spill_temp_1 = 8
        .loc    1 4 8
```

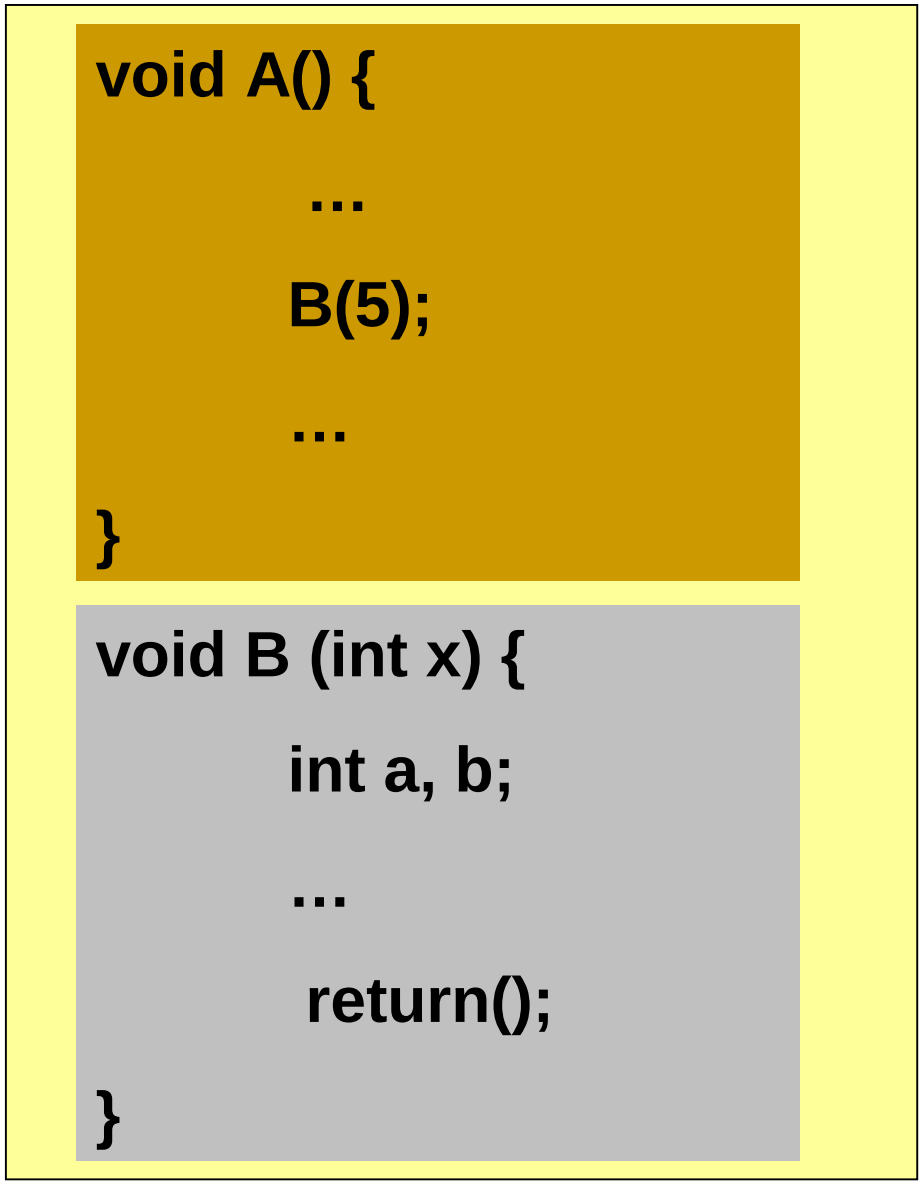
Assembly Representation..

```
#   1  #include<stdio.h>
#   2  #include<math.h>
#   3  float a[100];
#   4  main() {
.BB1.main:      # 0x0
.type   main, stt_func
        lui             $1,    %hi(%neg(%gp_rel(main)))      # [0]  main
        addiu           $sp,  $sp,      -16                  # [0]
        addiu           $1,    $1,      %lo(%neg(%gp_rel(main)))    # [1]  main
        sf              $gp,  0($sp)                        # [1]  gra_spill_temp_0
        addu            $gp,  $25,$1                        # [2]
        lw              $5,    %got_disp(a)($gp)            # [3]  a
        .loc    1 7 5
```

Example: Function Call and Return

Caller

Callee



Parameter

Function call

Return address

Local variables

Return

Example: Function Call and Return.

What must be done on a function call?

- Transfer control to start of function
- Remember return address
  - Where? In a General Purpose Register?

    No. The callee might have been compiled to use that register for its variables.

What must be done on a function return?

- Transfer control back to return address

Example: Function Call and Return..

What must be done on a function call?

- Transfer control to start of function
- Remember return address
  - Where? In a variable (main memory location)?
    No. That wouldn't work for nested or recursive function calls

What must be done on a function return?

- Transfer control back to return address

Example: Function Call and Return…

What must be done on a function call?

- Transfer control to start of function
- Remember return address
  - Where? On a stack (in main memory)?
    We could use the same stack for stack allocation of space for local variables and parameters of the function

What must be done on a function return?

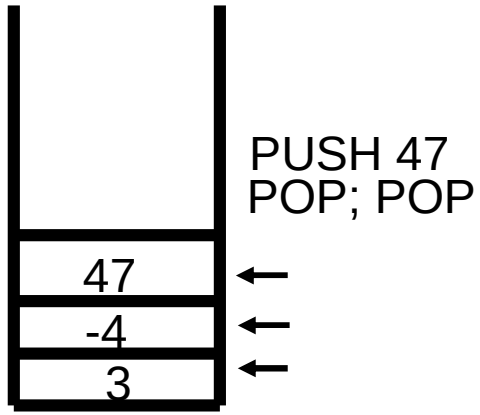
- Transfer control back to return address

Aside: What is a Stack?

- A data structure; like a stack of books
- Operations:

  Push: Insert onto top

  Pop: Delete from top



- Last In First Out (LIFO)
- SP: Stack Pointer, which keeps track of the current top of stack element

Example: Function Call and Return….

What must be done on a function call?

- Pass parameters on stack
- Transfer control to start of function
- Remember return address
  - Where? On a stack (in main memory)
- Allocate space for local variables on stack

What must be done on a function return?

- Pass return value (through stack)
- Clean up stack
- Transfer control back to return address

Problem: Separate Compilation

- Consider our simple example of compiling a C program in program.c that calls a math library function
- % gcc program.c
  - cc1 might use general purpose registers R3-R10 for the frequently used variables
  - But, what if these registers are used by the math function, which was compiled previously?
  - When the math function is called, the values in R3-R10 would be over written and therefore lost
  - Unless we save the values of those registers as part of the function call

Example: Function Call and Return

What must be done on a function call?

- Pass parameters on stack
- Transfer control to start of function
- Remember return address
  - Where? On a stack (in main memory)
- Save register values on stack
- Allocate space for local variables on stack

What must be done on a function return?

- Pass return value (through stack)
- Restore register values from the stack
- Clean up stack
- Transfer control back to return address