C Multi-module Programs

C Multi-module Programs

- Recall that we have developed a program previously that can reverse a string.

```c
/*  reverse.c  */
#include <stdio.h>

/*  Function prototype  */
void reverse (char* before, char* after);

/*****************************************/
int main()
{
  char str[100];   /* buffer to hold reversed string */
  reverse("cat",str);  /* reverse the string "cat"  */
  printf ("reverse(\"cat\") = %s\n", str);
  reverse("noon",str);
  printf ("reverse(\"noon\") = %s\n", str);
}

/*****************************************/
void reverse (char* before, char* after)
{
  int i,j,len;

  len = strlen(before);
  i=0;
  for (j=len-1; j>=0; j--)
  {
     after[i] = before[j];
     i++;
  }
  after[len] = NULL;
}
```

C Multi-module Programs

- This reverse function built previously cannot easily be used in other programs.
- Suppose that we wish to write a function that returns 1 if a string is a palindrome and 0 otherwise.
  - A palindrome is a string that reads the same forward and backward; for example, “noon” is a palindrome, but “nono” is not.
- We could use the reverse function to implement the palindrome function.

C Multi-module Programs

- One way to do this is to cut and paste reverse() into the palindrome program, but this is a poor technique for at least three reasons:
  - Performing a cut-and-paste operation is slow.
  - If we came up with a better piece of code for performing a reverse operation, we'd have to replace every copy of the old version with the new version, which is a maintenance nightmare.
  - Each copy of reverse() soaks up disk space.
- There is a better way to share functions.

Reusable Functions

- A better strategy for sharing reverse() is to
  - remove the function from the reverse program,
  - compile it separately,
  - and then *link* the resultant *object module* into whichever programs you wish to use it with.
- This technique avoids all three of the problems listed in the previous section and allows the function to be used in many different programs.
- Functions with this property are termed *reusable.*

Preparing a Reusable Function

- To prepare a reusable function, create a module that contains the source code of the function, together with a header file that contains the function's prototype.
- Then compile the source code module into an *object module* by using the **-c** option of **gcc**.
- An *object module* contains machine code, together with information, in the form of a symbol table, that allows the module to be combined with other object modules when an executable file is being created. Here are the listings of the new "reverse.c" and "reverse.h" files:

reverse.h

```
/* reverse.h */

void reverse (char *before, char *after);
    /* Declare but do not define this function */
```

Preparing a Reusable Function (con’t)

reverse.c

```
/* reverse.c */

#include <stdio.h>
#include “reverse.h”

/********************************************/

void reverse (char *before, char *after)

{
  int i;
  int j;
  int len;

  len = strlen (before);

  for (j = len – 1, i=0; j>= 0; j--,i++) /*Reverse loop*/
      after[i] = before[j];

   after[len] = NULL;  /* NULL terminate reversed string */
 }
```

Preparing a Reusable Function (con’t)

- Here’s a listing of a main program that uses reverse():

main1.c

```
/* main1.c */

#include <stdio.h>
#include “reverse.h” /*Contains the prototype of reverse)*/

/*********************************************/

int main ()

{
char str [100];

reverse (“cat”, str); /* Invoke external function */
printf (“reverse (\“cat\”) = %s\n”, str);
reverse (“noon”, str); /* Invoke external function */
printf (“reverse (\“noon\”) = %s\n”, str);
}
```

Compiling And Linking Modules Separately

- To compile each source code file separately, use the **-c** option of **gcc**. This creates a separate object module for each source code file, each with a “.o” suffix. The following commands are illustrative:

```
cuse93:> gcc -c reverse.c   ... compile reverse.c to reverse.o.
cuse93:> gcc -c main1.c     ... compile main1.c to main1.o.
cuse93:> ls -l reverse.o main1.o
-rw-r--r-- 1 glass   311 Jan 5 18:24 main1.o
-rw-r--r-- 1 glass   181 Jan 5 18:08 reverse.o
cuse93:> _
```

Compiling And Linking Modules Separately (con't)

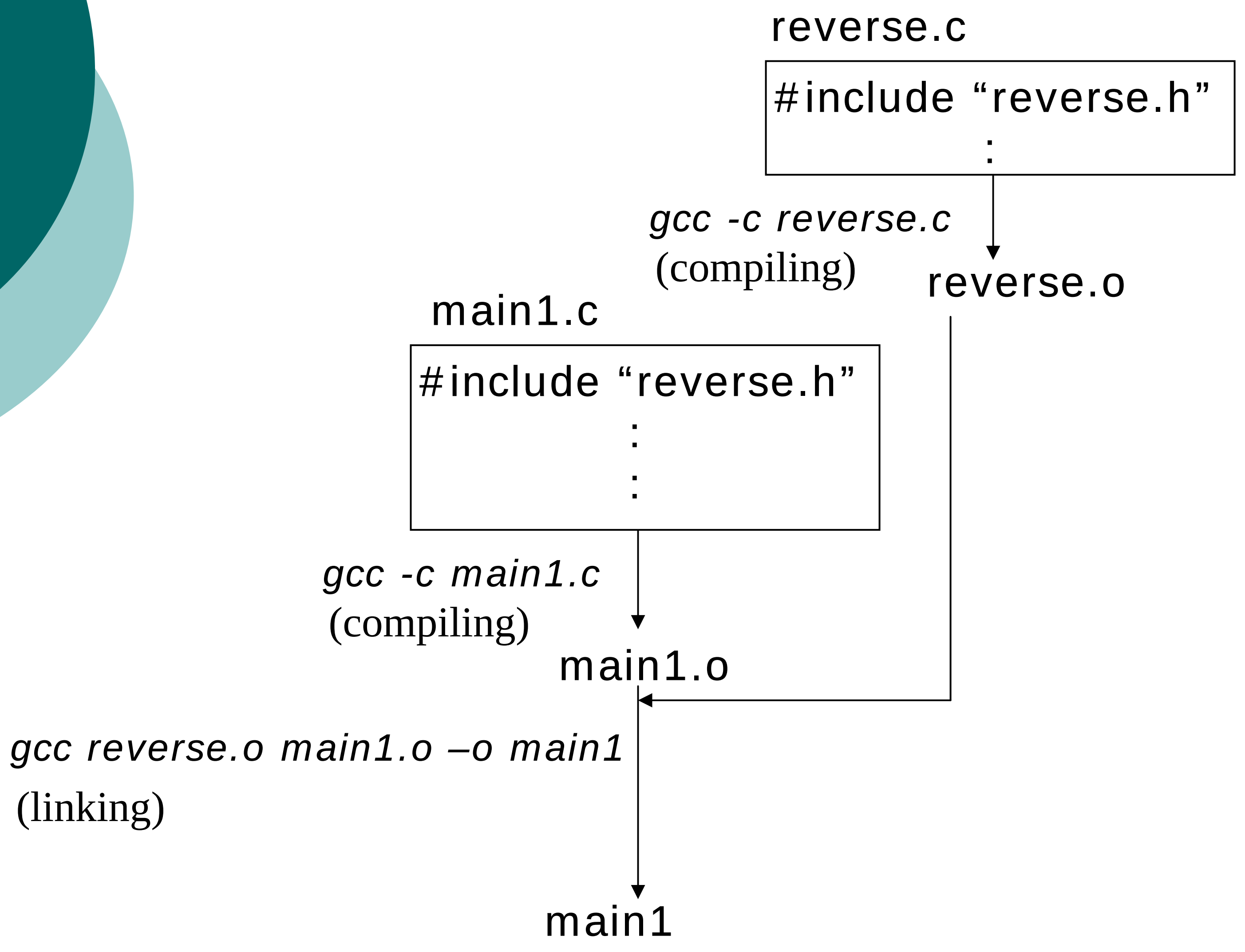
- Alternatively, you can list all of the source code files on one line:

```
cuse93:> gcc -c reverse.c main1.c      ... compile each .c file to .o file.
cuse93:> _
```

- To *link* them all together into an executable called "main1", list the names of all the object modules after the **gcc** command:

```
cuse93:> gcc reverse.o main1.o –o main1  ...link object modules.
cuse93:> ls -l main1
-rwxr—xr-x 1 glass      24576 Jan 5 18:25 main1*
cuse93:> ./main1                          ... run the executable.
reverse ("cat") = tac
reverse ("noon") = noon
cuse93:> _
```

Compiling And Linking Modules Separately – Facilitate Code Sharing



Modifying a Function

- Suppose that we modify the reverse function so that it prints out the value of some variables for debugging purpose.

Modifying a Function

reverse.c

```
/* reverse.c */

#include <stdio.h>
#include “reverse.h”

/********************************************/
void reverse (char* before, char* after)
{
  int i,j,len;

   len = strlen(before);
   i=0;
   for (j=len-1; j>=0; j--)
  {
      after[i] = before[j];
      i++;
          /* for debugging   */
          printf (“i=%d   j=%d\n”,i,j);
   }
   after[len] = NULL;
}
```

Modifying a Function

- There is no need to re-compile the main function main1.c since it has not been modified.
- Only the reverse function needs to be compiled. Then *link* all object modules and generate an executable (called "main1").

```
cuse93:> gcc -c reverse.c                          ... compile
cuse93:> gcc reverse.o main1.o –o main1    ...link object modules
```

- The output of running main1 is:

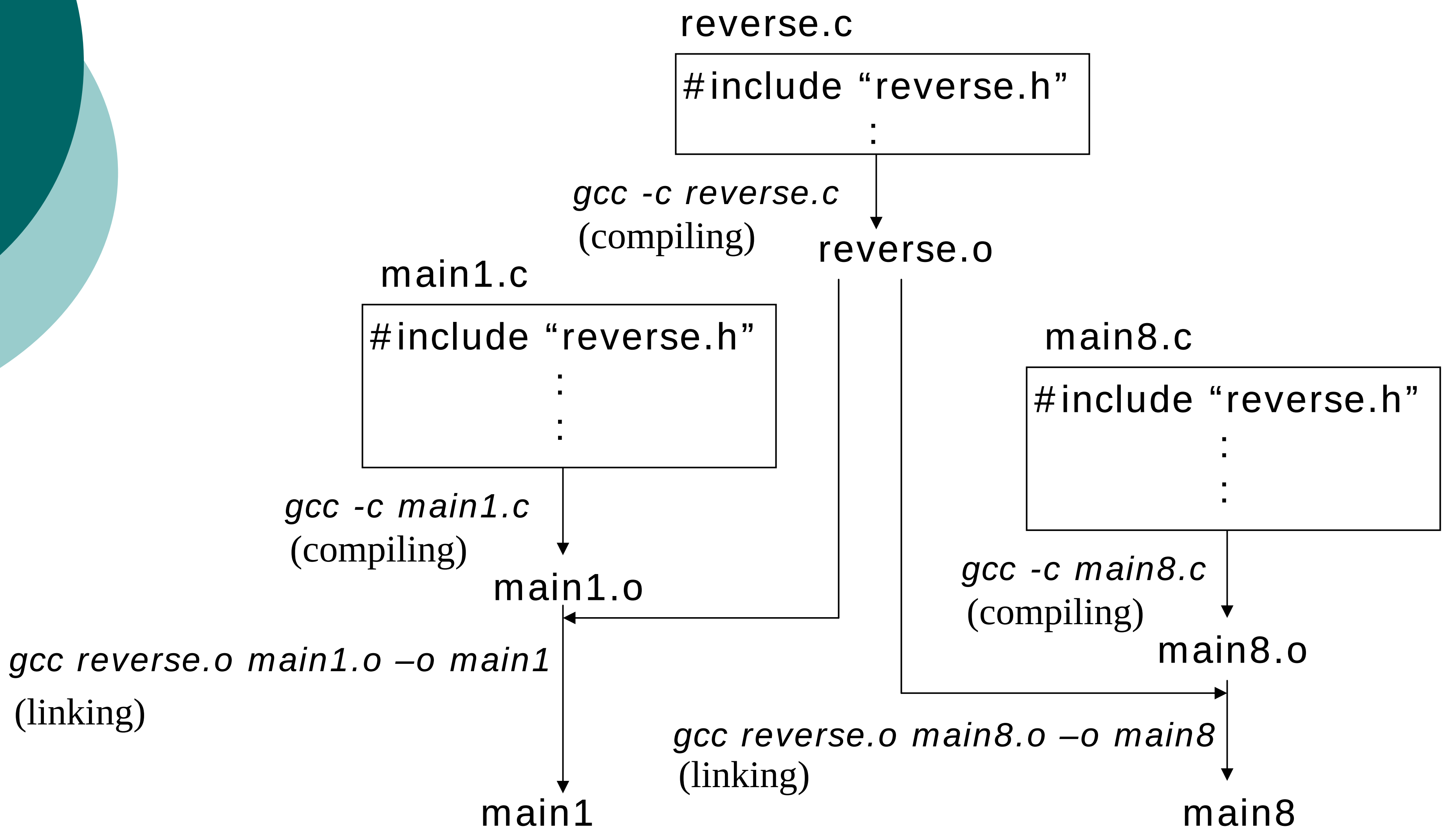
```
cuse93:> ./main1                              ... run the executable.
 i=1   j=2
i=2   j=1
i=3   j=0
reverse ("cat") = tac
i=1   j=3
        :
```

Re-using a Function

- The reverse function (module) can also be used by other programs such as main8.c and the compilation is as follows.

```
cuse93:> gcc -c main8.c                         ... compile
cuse93:> gcc reverse.o main8.o –o main8  ...link object modules
```

Compiling And Linking Modules Separately – Facilitate Code Sharing



Reusing The Reverse Function for Building Another Re-usable Function

- The reverse module can be used to build a program for testing palindrome

Reusing The Reverse Function for Building Another Re-usable Function

```
/* palindromall.c */
#include <stdio.h>
#include <string.h>
#include "reverse.h"

int palindrome (char *str) {
char reversedStr[100];

 reverse(str, reversedStr);
 return(strcmp(str,reversedStr) == 0);
}

int main() {
printf("palindrome(\"cat\") = %d\n", palindrome("cat"));
printf("palindrome(\"noon\") = %d\n", palindrome("noon"));
}
```

Reusing The Reverse Function for Building Another Re-usable Function

```
cuse93:> gcc –c  palindromeall.c      ... compile palindromeall.c to palindromeall.o
cuse93:> gcc reverse.o palindromeall.o -o palindromall   ... link them all
cuse93:> ./palindromeall                          ... run the program
palindrome (“cat”) = 0
palindrome (“noon”) = 1

cuse93:> _
```

Reusing The Reverse Function for Building Another Re-usable Function

- The way to combine the “reverse” and “palindromeall” modules is as we did before:
  - compile the object modules,
  - and then link them.
- We don’t have to recompile “reverse.c”, as it hasn’t changed since the “reverse.o” object file was created.

Reusing The Reverse Function for Building Another Re-usable Function

- The program can be further decomposed to multi-modules. Here are the header and source code listing of the palindrome function:

palindrome.h

```
/* palindrome.h */

int palindrome (char *str);
      /* Declare but do not define */
```

Reusing The Reverse Function for Building Another Re-usable Function

palindrome.c

```
/* palindrome.c */

#include “palindrome.h”
#include “reverse.h”
#include <string.h>

/*********************************************/

 int palindrome (char *str)

{
  char reversedStr [100];
  reverse (str, reversedStr); /* Reverse original */
  return (strcmp (str, reversedStr) ==0);
                                        /* Compare the two */
}
```

Reusing The Reverse Function for Building Another Re-usable Function

- The program “main2.c” that tests the palindrome function

```
/* main2.c */

#include <stdio.h>
#include “palindrome.h”

/********************************************/

int main ()

{
printf (“palindrome (\“cat\”) = %d\n”, palindrome (“cat”));
printf (“palindrome (\“noon\”) = %d\n”, palindrome(“noon”));
}
```

Reusing The Reverse Function for Building Another Re-usable Function

- The way to combine the “reverse”, “palindrome”, and “main2” modules is as we did before:
  - compile the object modules,
  - and then link them.
- We don’t have to recompile “reverse.c”, as it hasn’t changed since the “reverse.o” object file was created.

Reusing The Reverse Function for Building Another Re-usable Function

```
cuse93:> gcc –c  palindrome.c        ... compile palindrome.c to palindrome.o
cuse93:> gcc -c main2.c              ... compile main2.c to main2.o
cuse93:> gcc reverse.o palindrome.o main2.o -o main2   ... link them all.
cuse93:> ls  -l reverse.o palincdrome.o main2.o main2
-rwxr-xr-x 1 glass          24576 Jan 5 19:09 main2*
-rw-r--r--  1 glass           306 Jan 5 19:00 main2.o
-rw-r--r--  1 glass           189 Jan 5 18:59 palindrome.o
-rw-r--r--  1 glass           181 Jan 5 18:08 reverse.o
cuse93:> ./main2                             ... run the program.
palindrome (“cat”) = 0
palindrome (“noon”) = 1
cuse93:> _
```