Multi-module Programs

Motivating Example

- A Toyota! Race fast, safe car. A Toyota!

Motivating Example

- A Toyota! Race fast, safe car. A Toyota!
- A palindrome is a word or phrase that reads the same backward or forward
  - e.g. dad, noon
- Suppose we wish to develop a program (function) that can checks palindromes
- The previously developed function reverse() can be re-used.

C Multi-module Programs

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

```
1  /*  reverse.c  */
2  #include <stdio.h>
3  #include <string.h>
4  /*  Function prototype  */
5  void reverse (char before[], char after[]);
6
7  /*********************************************/
8  int main()
9  {
10   char str[100];   /* buffer to hold reversed string */
11   reverse("cat",str);  /* reverse the string "cat"  */
12   printf ("reverse(\"cat\") = %s\n", str);
13   reverse("noon",str);
14   printf ("reverse(\"noon\") = %s\n", str);
15 }
16
17 /*********************************************/
18 void reverse (char before[], char after[])
19 {
20   int i,j,len;
21
22   len = strlen(before);
23   i=0;
24   for (j=len-1; j>=0; j--)
25   {
26     after[i] = before[j];
27     i++;
28   }
29   after[len] = '\0';
30 }
```

C Multi-module Programs

- Suppose that we wish to write a function that returns 1 if a string is a palindrome and 0 otherwise.
- We attempt to re-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 function. Then, modify the code appropriately
  - The C library function:

  *int strcmp(const char *str1, const char *str2)*

  compares the string pointed to by str1 to the string pointed to by str2

  This function return a value that

  < 0 if str1 is less than str2

  > 0 if str1 is larger than str2

  = 0 if str1 is equal to str2
  - Recall that in C, *true* is represented by any numeric value not equal to 0 and *false* is represented by 0.
  - This program is *palindpoor.c*

C Multi-module Programs

- 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.

```
1   /*  palindpoor.c  */
2   #include <stdio.h>
3   #include <string.h>
4   int palindrome (char str[]);

5   int palindrome (char str[]) {
6      char reversedStr[100];
7      int i, j, len;
8      len = strlen(str);
9      i=0;
10     for (j = len-1; j>= 0; j--) {
11        reversedStr[i] = str[j];
12        i++;
13     }
14     reversedStr[len] = '\0';
15     return(strcmp(str,reversedStr) == 0);
16  }

17  int main() {
18     printf("palindrome(\"cat\") = %d\n", palindrome("cat"));
19     printf("palindrome(\"noon\") = %d\n", palindrome("noon"));
20     printf("palindrome(\"atoyotaracefastsafecaratoyota\") = %d\n",
            palindrome("atoyotaracefastsafecaratoyota"));
22  }
```

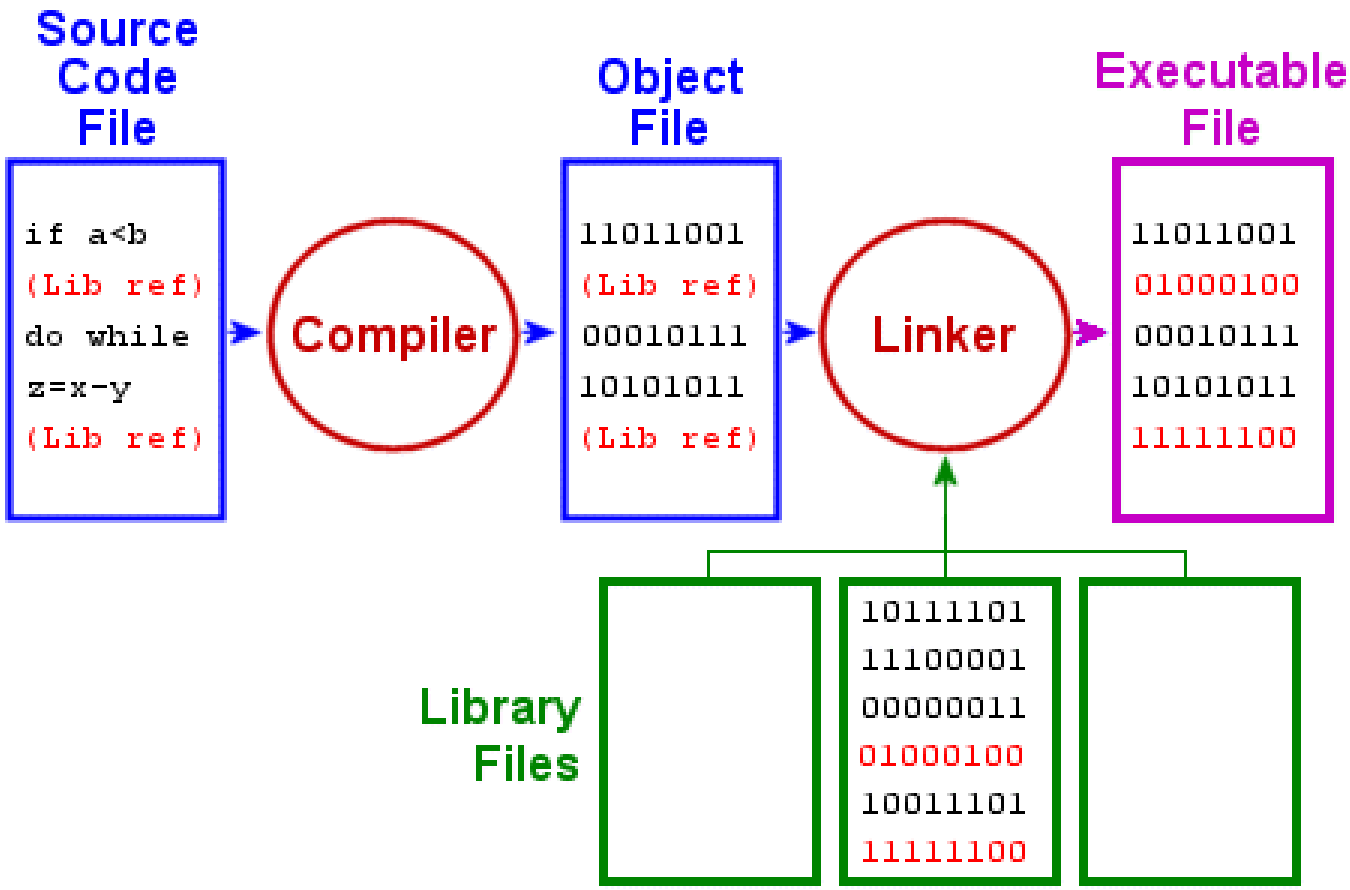
Reusable Functions

- 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.*

Compiling and Linking C programs

- C source files are human-readable text files.
- A *compiler* translates (compiles) source files into object modules (object-code files)
- 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.
- Then we use a system program called *linker* linking one or more object-code files to produce an executable file (an “executable”, or an “exe” file).
- Linkers can also link standard libraries and third-party libraries

Compiling and Linking C programs



Preparing a Reusable Function

- Returning to the C program reverse, suppose that we wish to prepare a reusable function. The strategy is to create a source code module that contains the source code of the function.
- Then compile the source code module into an *object module* by using the **-c** option of **gcc**.
- Here is the listing of the new “reverse0.c” file:

Preparing a Reusable Function (con't)

reverse0.c

```
1 /* reverse0.c */
2
3 #include <string.h>
4 void reverse (char before[], char after[]);
5
6 /*******************************************/
7
8 void reverse (char before[], char after[])
9
10 {
11   int i, j, len;
12
13   len = strlen (before);
14   i=0;
15   for (j = len – 1; j>= 0; j--)  {  /*Reverse loop*/
16       after[i] = before[j];
17       i++;
18    }
19   after[len] = '\0';  /* NULL terminate reversed string */
20  }
```

Preparing a Reusable Function (con't)

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

main0.c

```
1   /* main0.c */
2
3 #include <stdio.h>
4 void reverse(char before[], char after[]);
5
6 /*****************************************/
7
8 int main ()
9
10 {
11 char str [100];
12
13 reverse ("cat", str); /* Invoke external function */
14 printf ("reverse (\"cat\") = %s\n", str);
15 reverse ("noon", str); /* Invoke external function */
16 printf ("reverse (\"noon\") = %s\n", str);
17 }
```

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:

```
sepc92:> gcc -c reverse0.c  ...compile reverse0.c to reverse0.o.
sepc92:> gcc -c main0.c     ... compile main0.c to main0.o.
sepc92:> ls -l reverse0.o main0.o
-rw-r--r-- 1 glass   311 Jan 5 18:24 main0.o
-rw-r--r-- 1 glass   181 Jan 5 18:08 reverse0.o
sepc92:> _
```

Compiling And Linking Modules Separately (con't)

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

```
sepc92:> gcc -c reverse0.c main0.c      ... compile each .c file to .o file.
sepc92:> _
```

- To *link* them all together into an executable called "main0", list the names of all the object modules after the **gcc** command:

```
sepc92:> gcc reverse0.o main0.o –o main0  ...link object modules.
sepc92:> ls -l main0
-rwxr—xr-x 1 glass      24576 Jan 5 18:25 main0*
sepc92:> ./main0                                  ... run the executable.
reverse ("cat") = tac
reverse ("noon") = noon
sepc92:> _
```

Preparing a Reusable Function

- In software engineering, it is a good design to create a header file that contains the function's prototype.
- Then use #include to insert the header file at the beginning of the source files.
- For example, the header file can be called "reverse.h"

reverse.h

```
1 /* reverse.h */
2
3 void reverse (char before[], char after[]);
4      /* Declare but do not define this function */
```

- The source file "reverse.c" needs to be modified accordingly.

Preparing a Reusable Function (con't)

reverse.c

```
1 /* reverse.c */
2
3 #include <string.h>
4 #include "reverse.h"
5
6 /*********************************************/
7
8 void reverse (char before[], char after[])
9
10 {
11   int i, j, len;
12
13   len = strlen (before);
14   i=0;
15   for (j = len – 1; j>= 0; j--)  {  /*Reverse loop*/
16       after[i] = before[j];
17       i++;
18    }
19   after[len] = '\0';  /* NULL terminate reversed string */
20  }
```

Preparing a Reusable Function (con't)

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

main1.c

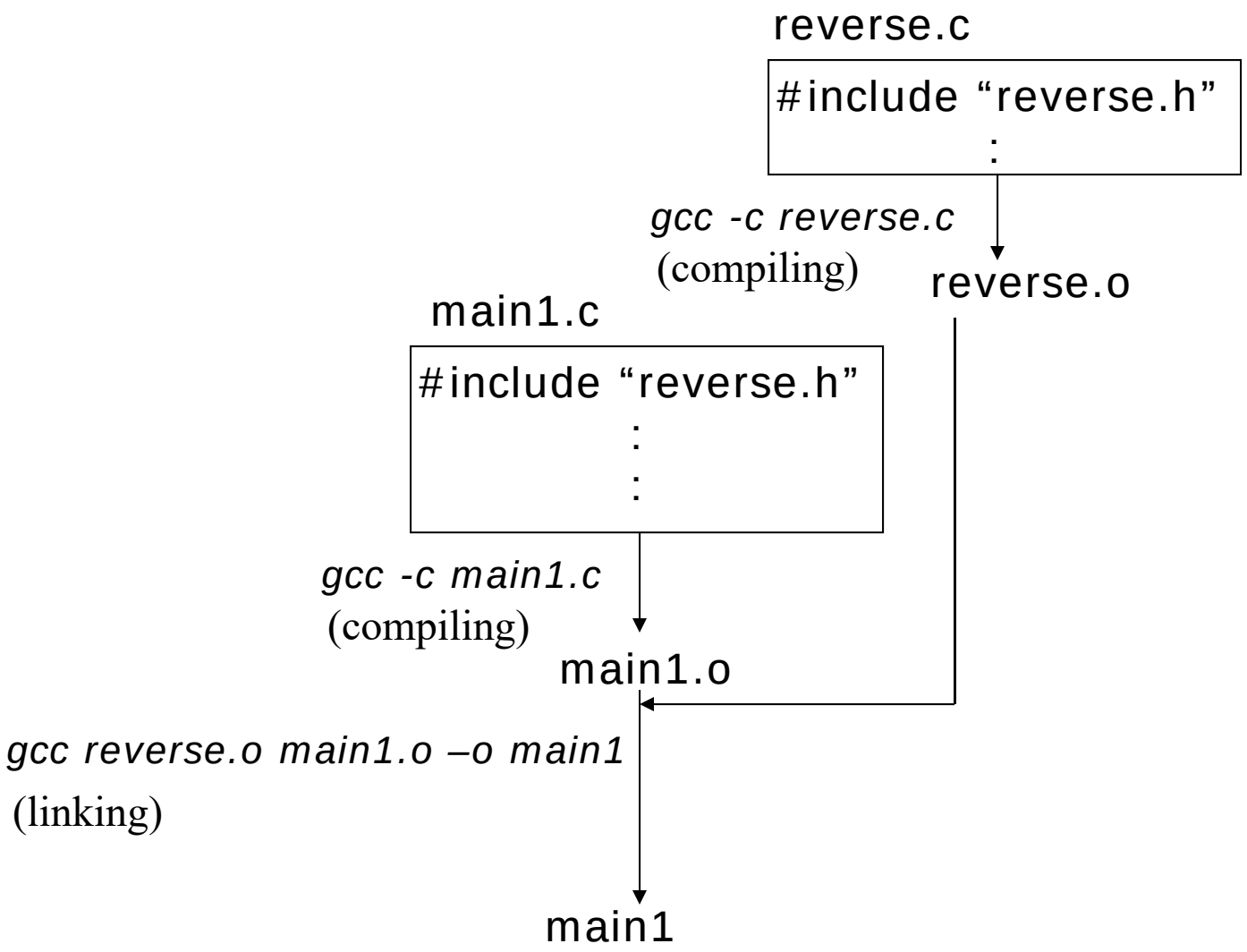
```
1   /* main1.c */
2
3 #include <stdio.h>
4 #include "reverse.h" /*Contains the prototype of reverse)*/
5
6 /*****************************************/
7
8 int main ()
9
10 {
11 char str [100];
12
13 reverse ("cat", str); /* Invoke external function */
14 printf ("reverse (\"cat\") = %s\n", str);
15 reverse ("noon", str); /* Invoke external function */
16 printf ("reverse (\"noon\") = %s\n", str);
17 }
```

Compiling And Linking Modules Separately

- Compile in a similar manner

```
sepc92:> gcc -c reverse.c   ... compile reverse.c to reverse.o.
sepc92:> gcc -c main1.c     ... compile main1.c to main1.o.
sepc92:> gcc reverse.o main1.o –o main1
sepc92:> _
```

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

```
1 /* reverse.c */
2 #include <stdio.h>      /* need this header file for printf  */
3 #include <string.h>
4 #include “reverse.h”
5
6 /*********************************************/
7 void reverse (char before[], char after[])
8 {
9    int i,j,len;
10
11   len = strlen(before);
12   i=0;
13   for (j=len-1; j>=0; j--)
14  {
15      after[i] = before[j];
16      i++;
17          /* for illustration of modifying the algorithm   */
18          printf (“i=%d   j=%d\n”,i,j);
19   }
20   after[len] = ‘\0’;
21 }
```

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").

  sepc92:> *gcc -c reverse.c* … compile

  sepc92:> *gcc reverse.o main1.o –o main1* …link object modules

- The output of running main1 is:

  sepc92:> *./main1* … run the executable.

  i=1 j=2

  i=2 j=1

  i=3 j=0

  reverse ("cat") = tac

  i=1 j=3

  :

Reusing a Function

- Here's a listing of another program that uses reverse():

main8.c

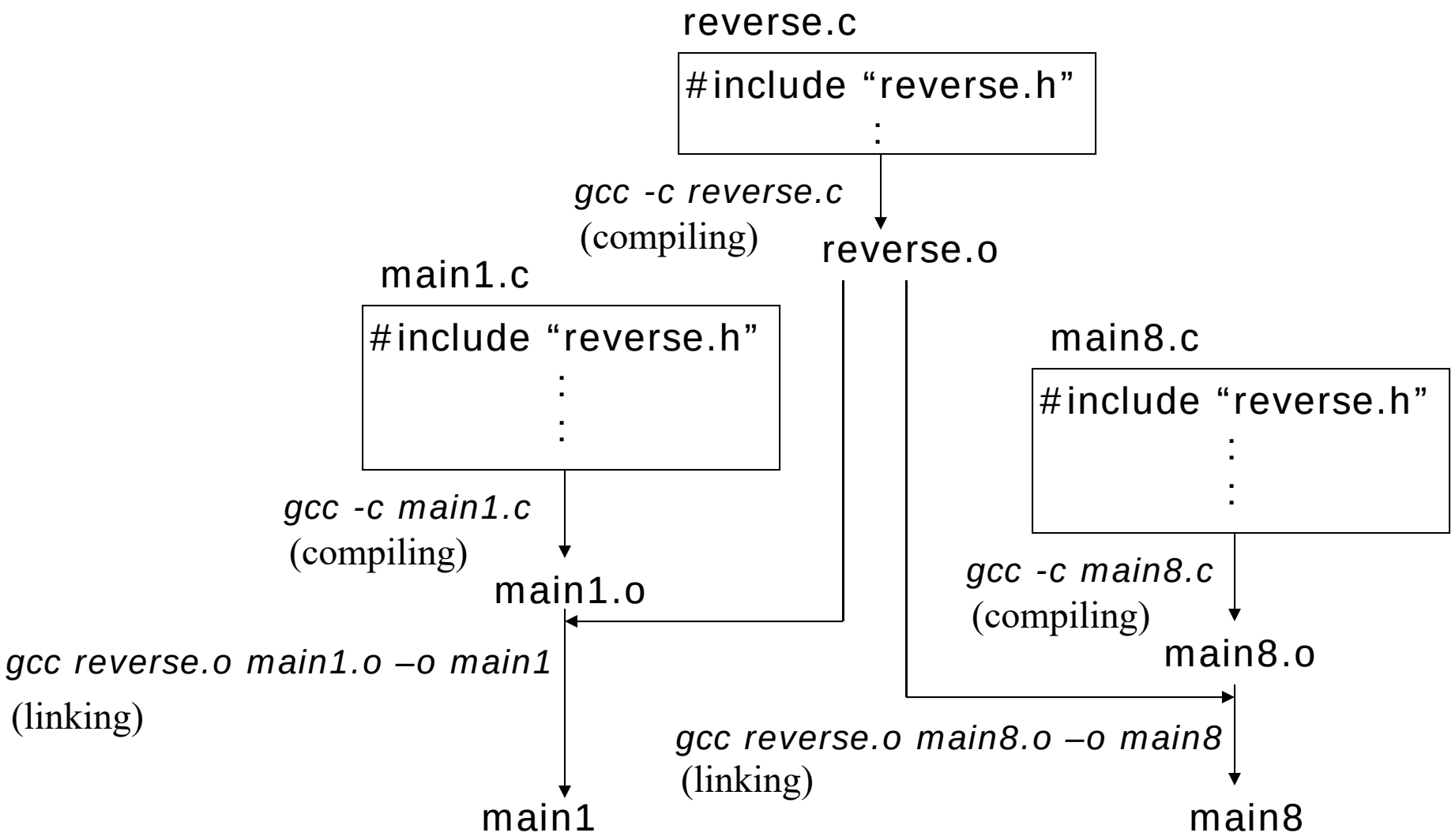
```
1   /* main8.c */
2 #include <stdio.h>
3 #include <string.h>
4 #include “reverse.h” /*Contains the prototype of reverse)*/
5
6 /*****************************************/
7
8 int main ()
9 {
10    char person[100];
11    reverse ("tom", person); /* Invoke external function */
12    strcat(person, " Berth");
13    printf ("person = (%s)\n", person);
14 }
```

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.

sepc92:> *gcc -c main8.c* ... compile
sepc92:> *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 Function

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

Reusing The Reverse Function for Building Another Function

```
/*  palindromall.c  */
#include <stdio.h>
#include <string.h>
#include "reverse.h“
int palindrome (char str[]);

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"));
    printf("palindrome(\"atoyotaracefastsafecaratoyota\") = %d\n",
         palindrome("atoyotaracefastsafecaratoyota"));
}
```

Reusing The Reverse Function for Building Another Function

sepc92:> *gcc –c palindromeall.c* … compile palindromeall.c to palindromeall.o

sepc92:> *gcc reverse.o palindromeall.o -o palindromall* … link them all

sepc92:> *./palindromeall* … run the program

palindrome(“cat”) = 0
palindrome(“noon”) = 1
palindrome("atoyotaracefastsafecaratoyota") = 1

sepc92:> _

Reusing The Reverse Function for Building Another 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

```
1 /* palindrome.h */
2
3 int palindrome (char str[]);
4       /* Declare but do not define */
```

Reusing The Reverse Function for Building Another Re-usable Function

palindrome.c

```
1 /* palindrome.c */
2
3 #include “palindrome.h”
4 #include “reverse.h”
5 #include <string.h>
6
7 /*********************************************/
8
9  int palindrome (char str[])
10
11 {
12   char reversedStr [100];
13   reverse (str, reversedStr); /* Reverse original */
14   return (strcmp (str, reversedStr) ==0);
15                                        /* Compare the two */
16 }
```

Reusing The Reverse Function for Building Another Re-usable Function

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

```
1 /* main2.c */
2
3 #include <stdio.h>
4 #include “palindrome.h”
5
6 /*************************************/
7
8 int main ()
9
10 {
11 printf(“palindrome(\“cat\”) = %d\n”, palindrome (“cat”));
12 printf(“palindrome(\“noon\”) = %d\n”, palindrome(“noon”));
13 printf("palindrome(\"atoyotaracefastsafecaratoyota\") = %d\n",
        palindrome("atoyotaracefastsafecaratoyota"));
15 }
```

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

```
sepc92:> gcc –c  palindrome.c      ... compile palindrome.c to palindrome.o

sepc92:> gcc -c main2.c            ... compile main2.c to main2.o

sepc92:> gcc reverse.o palindrome.o main2.o -o main2   ... link them all.
sepc92:> ls  -l reverse.o palindrome.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
sepc92:> ./main2                        ... run the program.
palindrome(“cat”) = 0
palindrome(“noon”) = 1
palindrome("atoyotaracefastsafecaratoyota") = 1
sepc92:> _
```