KU LEUVEN

Introduction to C

Multifile, etc.

1

Working with multiple files

- Compiling
- Function prototypes / Function headers
- Scope

ICTS KU LEUVEN

2

Multiple files

- When writing large programs: divide programs up into modules.
  - separate source files.
  - main() would be in one file
  - the others files will contain functions.
- Modules can be shared amongst many programs by simply including the modules at compilation time.

ICTS KU LEUVEN

Multiple files

- Some rules
  - There is only one definition of the `main()` function in the program.
  - Any user-defined function must be completely defined in one file.
  - The file from where the function is called (but in which it is not defined) must include a prototype of the called function

KU LEUVEN

Compiling with multiple sources

- If the source code is in several files, say "file1.c" and "file2.c", then they can be compiled into an executable program named "myprog" using the following command:

```
gcc file1.c file2.c -o myprog
```

KU LEUVEN

Compiling with multiple sources

- The same result can be achieved using the following three commands:

```
gcc -c file1.c
gcc -c file2.c
gcc file1.o file2.o -o myprog
```

- The advantage of the second method is that it compiles each of the source files separately.
  If "file1.c" was modified, then the following commands would correctly update "myprog".

```
gcc -c file1.c
gcc file1.o file2.o -o myprog
```

- Tip: use `make` to automate the process

KU LEUVEN

```c
#include <stdio.h>

/*
calculate_all.c

perform some calculations on 2 integer numbers

*/

int avg( int x, int y );
int largest( int x, int y);


int main()
{
  int a = 10;
  int b = 20;
  int r1, r2;
  r1 = avg( a, b );
  r2 = largest( a, b);
  printf( "average of %d and %d is %d\n", a, b, r1 );
  printf( "largest of %d and %d is %d\n", a, b, r2 );
  return  0;
}

int avg( int x, int y )
{
  int sum = x + y;
  return sum / 2;
}

int largest( int x, int y )
{
  int large;
  if (x > y) {
  large = x;}
  else {
  large = y;}
  return large;
}
```

```
frankvp@CRD-L-08004:.../more$ gcc calculate_all.c -o calculate_all
frankvp@CRD-L-08004:.../more$ ./calculate_all
average of 10 and 20 is 15
largest of 10 and 20 is 20
frankvp@CRD-L-08004:.../more$
```

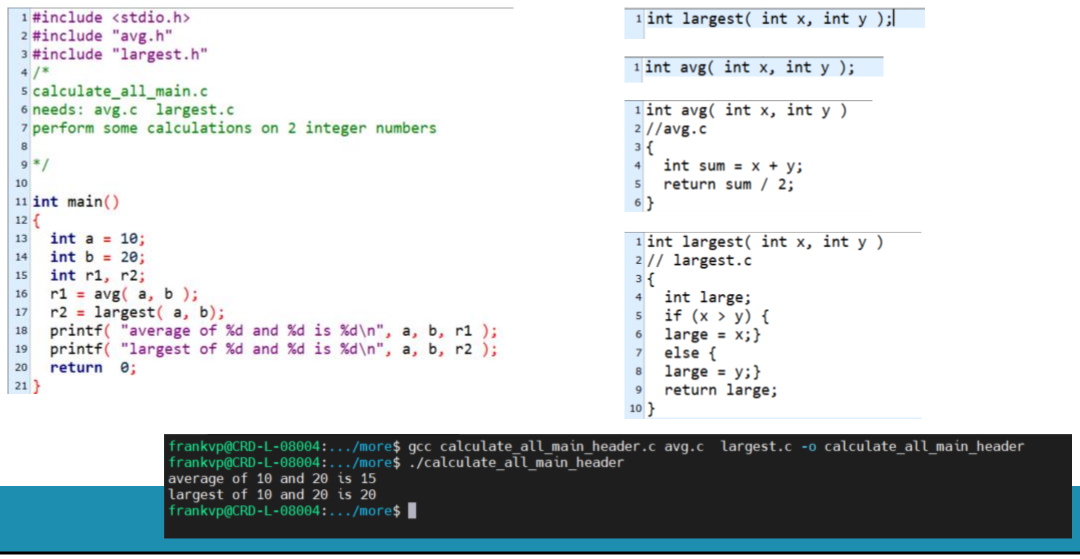
```c
#include <stdio.h>

/*
calculate_all_main.c
needs: avg.c  largest.c
perform some calculations on 2 integer numbers

*/

int avg( int x, int y );
int largest( int x, int y);


int main()
{
  int a = 10;
  int b = 20;
  int r1, r2;
  r1 = avg( a, b );
  r2 = largest( a, b);
  printf( "average of %d and %d is %d\n", a, b, r1 );
  printf( "largest of %d and %d is %d\n", a, b, r2 );
  return  0;
}
```

Prototypes (*declarations*) are used when the compiler must be informed about a function

```c
int avg( int x, int y )
//avg.c
{
  int sum = x + y;
  return sum / 2;
}
```

```c
int largest( int x, int y )
// largest.c
{
  int large;
  if (x > y) {
  large = x;}
  else {
  large = y;}
  return large;
}
```

```
frankvp@CRD-L-08004:.../more$ gcc calculate_all_main.c avg.c  largest.c -o calculate_all_main
frankvp@CRD-L-08004:.../more$ ./calculate_all_main
average of 10 and 20 is 15
largest of 10 and 20 is 20
frankvp@CRD-L-08004:.../more$
```



```c
#include <stdio.h>
#include "avg.h"
#include "largest.h"
/*
calculate_all_main.c
needs: avg.c  largest.c
perform some calculations on 2 integer numbers

*/

int main()
{
  int a = 10;
  int b = 20;
  int r1, r2;
  r1 = avg( a, b );
  r2 = largest( a, b);
  printf( "average of %d and %d is %d\n", a, b, r1 );
  printf( "largest of %d and %d is %d\n", a, b, r2 );
  return  0;
}
```

```c
int largest( int x, int y );
```

```c
int avg( int x, int y );
```

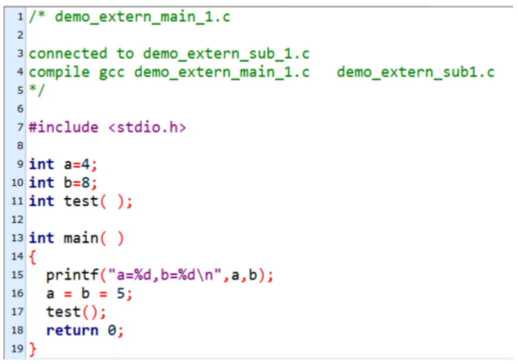
```c
int avg( int x, int y )
//avg.c
{
  int sum = x + y;
  return sum / 2;
}
```

```c
int largest( int x, int y )
// largest.c
{
  int large;
  if (x > y) {
  large = x;}
  else {
  large = y;}
  return large;
}
```

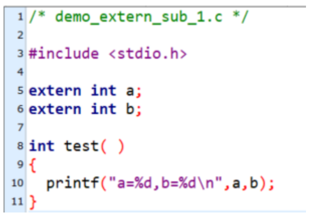
```
frankvp@CRD-L-08004:.../more$ gcc calculate_all_main_header.c avg.c  largest.c -o calculate_all_main_header
frankvp@CRD-L-08004:.../more$ ./calculate_all_main_header
average of 10 and 20 is 15
largest of 10 and 20 is 20
frankvp@CRD-L-08004:.../more$
```

Scope

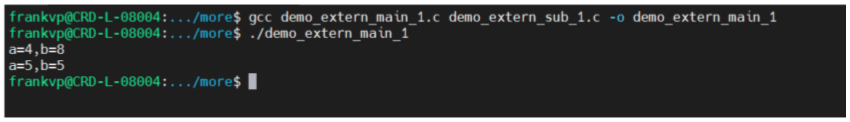
- **Local** (automatic) variables are only recognized in the function where they are defined. Once that function exits, that variable no longer exists and the memory that was allocated to it is returned to the free memory stack.
- **Global** variables: global variables are defined outside of any function. These variables must be declared in any other file where they are to be recognized as being the same variable. This is done by preceding the declarations in the other files with the identifier `extern`
  - Each global variable must be *defined* inside *exactly* one of the files
  - Each global variable must be *declared* inside *every* C program files
- What if global variable is only to be recognized as such only in the file where it is defined? The identifier `static` is used in the definition of the global variable to make it only accessible to functions within the same file where it is defined.



```c
/* demo_extern_main_1.c

connected to demo_extern_sub_1.c
compile gcc demo_extern_main_1.c   demo_extern_sub1.c
*/

#include <stdio.h>

int a=4;
int b=8;
int test( );

int main( )
{
  printf("a=%d,b=%d\n",a,b);
  a = b = 5;
  test();
  return 0;
}
```

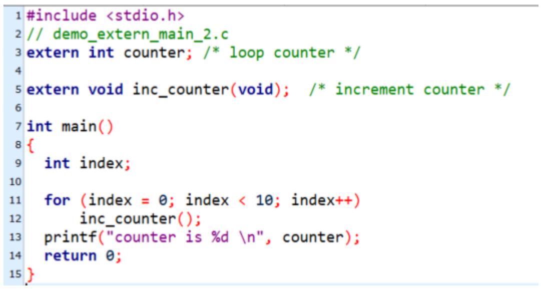


```c
/* demo_extern_sub_1.c */

#include <stdio.h>

extern int a;
extern int b;

int test( )
{
  printf("a=%d,b=%d\n",a,b);
}
```

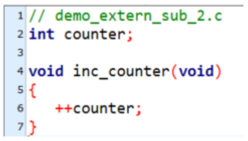


```
frankvp@CRD-L-08004:.../more$ gcc demo_extern_main_1.c demo_extern_sub_1.c -o demo_extern_main_1
frankvp@CRD-L-08004:.../more$ ./demo_extern_main_1
a=4,b=8
a=5,b=5
frankvp@CRD-L-08004:.../more$ █
```

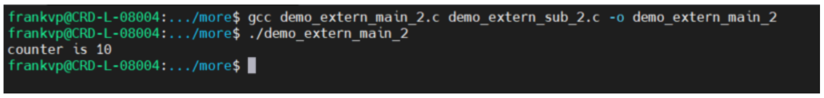
ICTS KU LEUVEN



```c
#include <stdio.h>
// demo_extern_main_2.c
extern int counter; /* loop counter */

extern void inc_counter(void);  /* increment counter */

int main()
{
  int index;

  for (index = 0; index < 10; index++)
      inc_counter();
  printf("counter is %d \n", counter);
  return 0;
}
```



```c
// demo_extern_sub_2.c
int counter;

void inc_counter(void)
{
   ++counter;
}
```



```
frankvp@CRD-L-08004:.../more$ gcc demo_extern_main_2.c demo_extern_sub_2.c -o demo_extern_main_2
frankvp@CRD-L-08004:.../more$ ./demo_extern_main_2
counter is 10
frankvp@CRD-L-08004:.../more$ █
```

ICTS KU LEUVEN

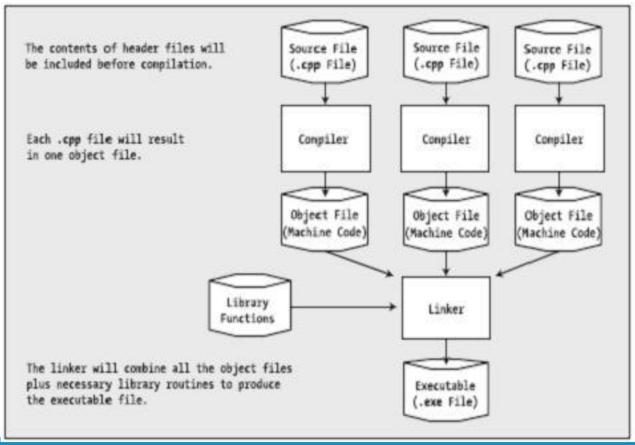
External Declaration

- Declaration of external variables or functions
  - The keyword **`extern`** is used to declare a variable in one file that is defined in another
  - The keyword **`extern`** is optional for function declarations since they are external by default
- Declaration versus definition
  - A variable or function must have only one *definition* in entire program
    - Definition allocates storage
  - A variable or function may have multiple *declarations*; one in every source file that uses it
    - Declaration permits linkage

Using External Variables

- Should be avoided in general.
  - Can almost always create better designs passing local variables via function arguments
- Tend to tie functions together (induce dependencies)
- Breaks modularity.

Compilation Model


The contents of header files will be included before compilation.
Each .cpp file will result in one object file.
The linker will combine all the object files plus necessary library routines to produce the executable file.
Source File (.cpp File)
Source File (.cpp File)
Source File (.cpp File)
Compiler
Compiler
Compiler
Object File (Machine Code)
Object File (Machine Code)
Object File (Machine Code)
Library Functions
Linker
Executable (.exe File)

KU LEUVEN

External Scope

```
/* File one.c */
#include<stdio.h>
int a;
int main(void)
{
   int b;
   a = 2;
   printf(“a = %d”, a);
   b = myfun();
   printf(“b = %d”, b);
}
```

```
/* File two.c */

/* When this file is linked with one.c,
  functions of this file can access a
*/
extern int a;
int myfun()
{
   a = 2;
   return a * 10;
}
```

KU LEUVEN