CS 2211
System Programming

Part Nine (a): Header Files

FILES

main.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
      int x,y,d,r;
      float rT;

      x=9;
      y=2;

      if (denominator < 1)
         return(-1);
      d=numerator/denominator;
      r=numerator%denominator;
      rT = ( (float)numerator/ (float)denominator);

      printf("%d/%d = %d with %d remainder\n",x,y,d,r);
      printf("%d/%d = %f ",x,rT);
}
```

FILES

main.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
      int x,y,d,r;
      float rT;

      x=9;
      y=2;
      rT = division(x,y,&d,&r);
      printf("%d/%d = %d with %d remainder\n",x,y,d,r);
      printf("x=%d\n",x);
}

float division(int numerator, int denominator,
               int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(-1);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    return( (float)numerator/ (float)denominator);
}
```

FILES

main.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
      int x,y,d,r;
      float rT;

      x=9;
      y=2;
      rT = division(x,y,&d,&r);
      printf("%d/%d = %d with %d remainder\n",x,y,d,r);
      printf("x=%d\n",x);
}

float division(int numerator, int denomi
              int *dividend, int *remaind
{
    printf("address stored in dividend:
    printf("address stored in remainder:
    if (denominator < 1)
        return(-1);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    return( (float)numerator/ (float)denominator);
}
```

goal:
break this down into a **main** program
and all other **functions** in separate files

FILES

main.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
      int x,y,d,r;
      float rT;

      x=9;
      y=2;
      rT = division(x,y,&d,&r);
      printf("%d/%d = %d with %d remainder\n",x,y,d,r);
      printf("x=%d\n",x);
}
```

functions.c

```
float division(int numerator, int denominator,
               int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(-1);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    return( (float)numerator/ (float)denominator);
}
```

FILES

main.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
      int x,y,d,r;
      float rT;

      x=9;
      y=2;
      rT = division(x,y,&d,&r);
      printf("%d/%d = %d with %d
      printf("x=%d\n",x);
}
```

error:
code will compile and run
but !
the answer will be incorrect.

preprocessor does not 'know' the return type of the function division
so
it assigns the default of type int

running this code will return a value of
4.0000000
for rT (based on integer not floating point)

```
float division(int numerator, in
               int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(-1);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    return( (float)numerator/ (float)denominator);
}
```

FILES

main.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
      int x,y,d,r;
      float rT;

      x=9;
      y=2;
      rT = division(x,y,&d,&r);
      printf("%d/%d = %d with %d
      printf("x=%d\n",x);
}
```

error:
code will compile and run
but !
the answer will be incorrect.

preprocessor does not 'know' the return type of the function division
so
it assigns the default of type int

running this code will return a value of
4.0000000
for rT (based on integer not floating point)

```
float division(int numerator, in
               int *dividend, int *remainder)
{
    printf("address stored in di
    printf("address stored in re
    if (denominator < 1)
        return(-1);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    return( (float)numerator/ (float)denominator);
}
```

must 'prototype' the function so the preprocessor knows what to expect as a return type

FILES

main.c

```
#include <stdio.h>
float division(int , int , int* , int*);

int main(int argc, char *argv[])
{
      int x,y,d,r;
      float rT;

      x=9;
      y=2;
      rT = division(x,y,&d,&r);
      printf("%d/%d = %d with %d remainder\n",x,y,d,r);
      printf("x=%d\n",x);
}
```

warning:
the prototype MUST end with a semicolon

functions.c

```
float division(int numerator, int denominator,
               int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(-1);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    return( (float)numerator/ (float)denominator);
}
```

FILES

main.c

```
#include <stdio.h>
float division(int , int , int* , int*);

int main(int argc, char *argv[])
{
      int x,y,d,r;
      float rT;

      x=9;
      y=2;
      rT = division(x,y,&d,
      printf("%d/%d = %d wi
      printf("x=%d\n",x);
}
```

```
float division(int numerat
               int *dividend
{
    printf("address stored
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(-1);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    return( (float)numerator/ (float)denominator);
}
```

the list of prototype can get very large and confusing.

solution:
create a header file
and in this file place

```
prototypes
macro definitions (#define)
data type definitions
variable definitions
```

FILES

main.c

```c
#include <stdio.h>
#include "prog1.h"

int main(int argc, char *argv[])
{
      int x,y,d,r;
      // float rT;    // make this a global variable

      x=9;
      y=2;
      rT = division(x,y,&d,&r);
      printf("%d/%d = %d with %d remainder\n",x,y,d,r);
      printf("x=%d\n",x);
}
```

prog1.h

```c
#ifndef PROG1_H_INCLUDED
  #define PROG1_H_INCLUDED

  float rT;

  float division(int , int , int* , int*);

#endif // PROG1_H_INCLUDED
```

.c

```c
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    return( (float)numerator/ (float)denominator);
}
```

FILES

main.c

```c
#include <stdio.h>
#include "prog1.h"

int main(int argc, char *argv[])
{
      int d,r;
      // float rT;     // make this a global variable
      NUMB x, y;
      x=9;
      y=2;
      rT = division(x,y,&d,&r);
      printf("%d/%d = %d with %d remainder\n",x,y,d,r);
      printf("x=%d\n",x);
}
```

prog1.h

```c
#ifndef PROG1_H_INCLUDED
  #define PROG1_H_INCLUDED

  #define NUMB int

  float rT;

  float division(int , int , int* , int*);

#endif // PROG1_H_INCLUDED
```

.c

```c
    return( (float)numerator/ (float)denominator);
}
```

FILES

main.c

```
#include <stdio.h>
#include "mainHeader.h"

int main()  {
     NUMB x, y, d, r;
     x=9;
     y=2;
     printf("address of d: %p\n",&d);
     printf("address of r: %p\n",&r);
     rT = division(x,y,&d,&r);
     printf("%d/%d = %d with %d remainder\n",x,y,d,r);
     printf("x=%d\n",x);
     printf("rT=%f\n",rT);
}
```

functions.h

```
float division(int numerator, int denominator,
               int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(-1);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    return( (float)numerator/ (float)denominator);
}
```

FILES

main.c

```
#include <stdio.h>
#include "mainHeader.h"

int main()  {
      NUMB x, y, d, r;
      x=9;
      y=2;
      printf("address of d: %p\n",&d);
      printf("address of r: %p\n",&r);
      rT = division(x,y,&d,&r);
      printf("%d/%d = %d with %d remainder\n",x,y,d,r);
      printf("x=%d\n",x);
      printf("rT=%f\n",rT);
}
```

functions.h

```
float division(int numerator, int denominator,
}
```

definitionsHeader.h

```
#ifndef dH
#define dH

  #define  NUMB int

  float rT;
  float division(int , int , int* , int*);

#endif // dH
```

FILES

main.c

```
#include <stdio.h>
#include "mainHeader.h"

int main()  {
     NUMB x, y, d, r;
     x=9;
```

mainHeader.h

```
#ifndef mH
#define mH

#include "definitionsHeader.h"
#include "functions.h"

#endif // mH
```

```
}
```

functions.h

```
float division(int numerator, int denominator,
```

definitionsHeader.h

```
#ifndef dH
#define dH

  #define  NUMB int

  float rT;
  float division(int , int , int* , int*);

#endif // dH
```

```
}
```

mainHeader.h

```
#ifndef mH
#define mH

#include "definitionsHeader.h"
#include "functions.h"

#endif // mH
```

definitionsHeader.h

```
#ifndef dH
#define dH

  #define  NUMB int

  float rT;
  float division(int , int , int* , int*);

#endif // dH
```

functions.h

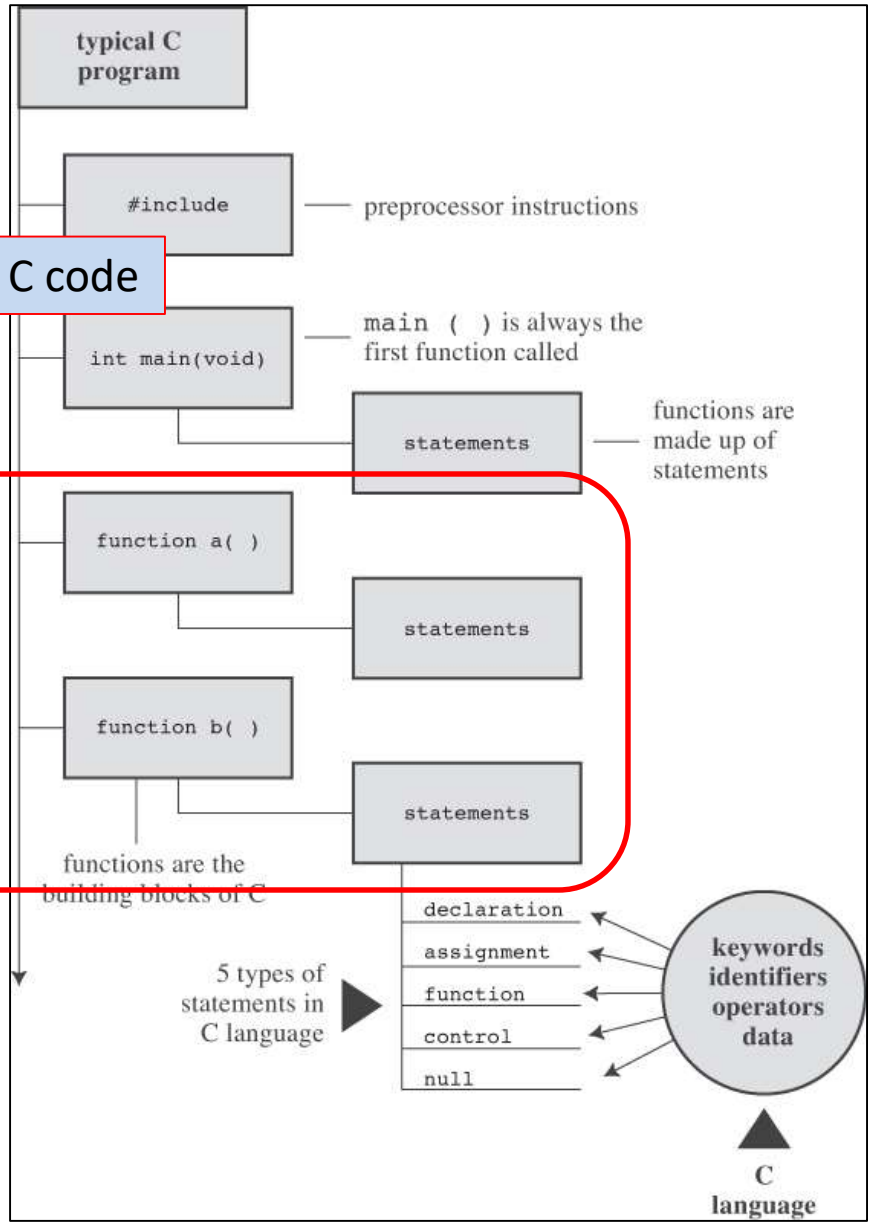
```
float division(int numerator, int denominator, int *dividend, int *remainder)  {
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(-1);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    return( (float)numerator/ (float)denominator);
}
```

main.c

```
#include <stdio.h>
#include "mainHeader.h"

int main()  {
     NUMB x, y, d, r;
     x=9;
     y=2;
     printf("address of d: %p\n",&d);
     printf("address of r: %p\n",&r);
     rT = division(x,y,&d,&r);
     printf("%d/%d = %d with %d remainder\n",x,y,d,r);
     printf("x=%d\n",x);
     printf("rT=%f\n",rT);
}
```

LAYERS of a C PROGRAM

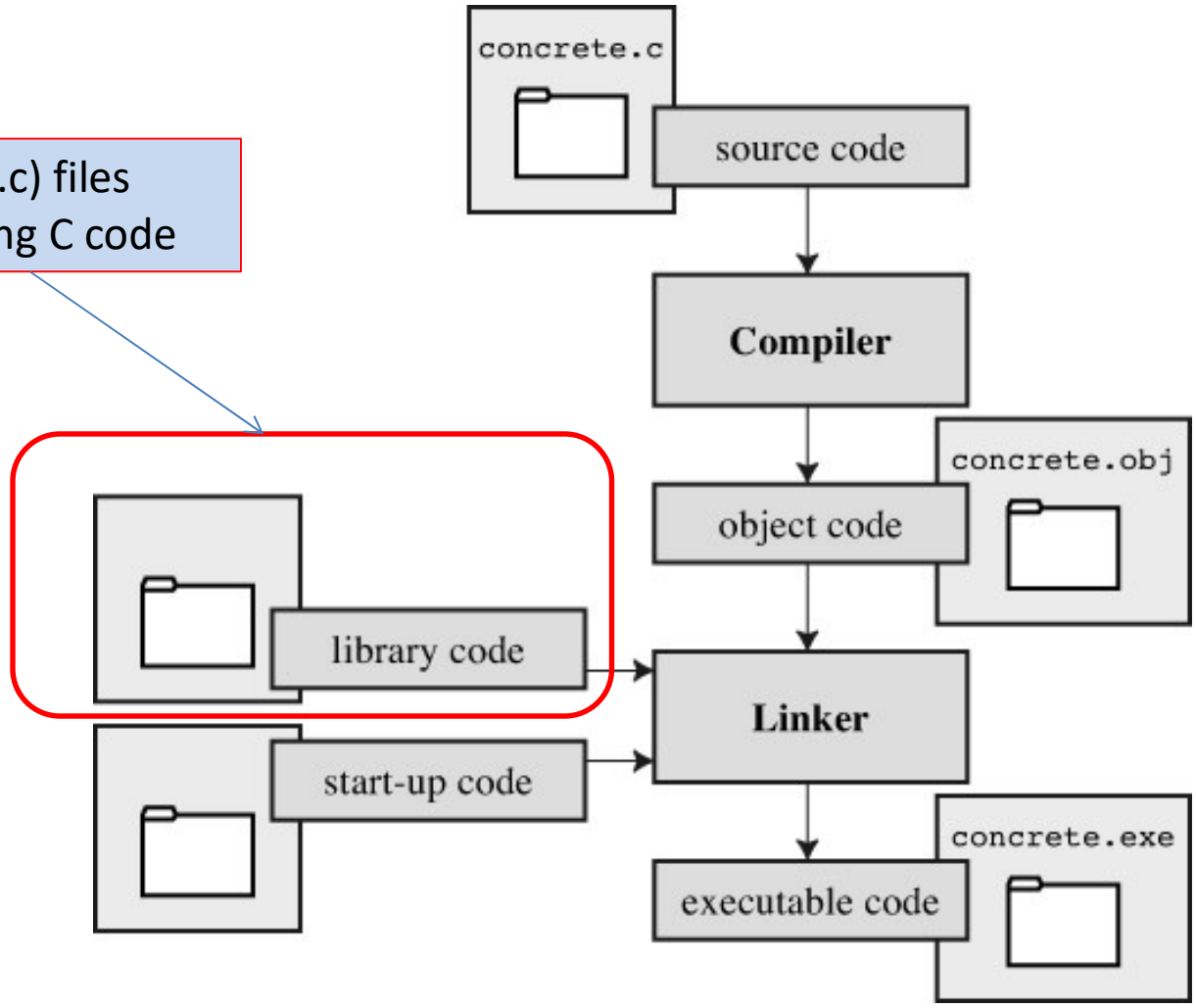


header files containing C code

compiled

COMPILING a C PROGRAM

code (xx.c) files containing C code



object code

Header Files in C

END OF SECTION

CS 2211
System Programming

Part Nine (b): Pointers to Functions

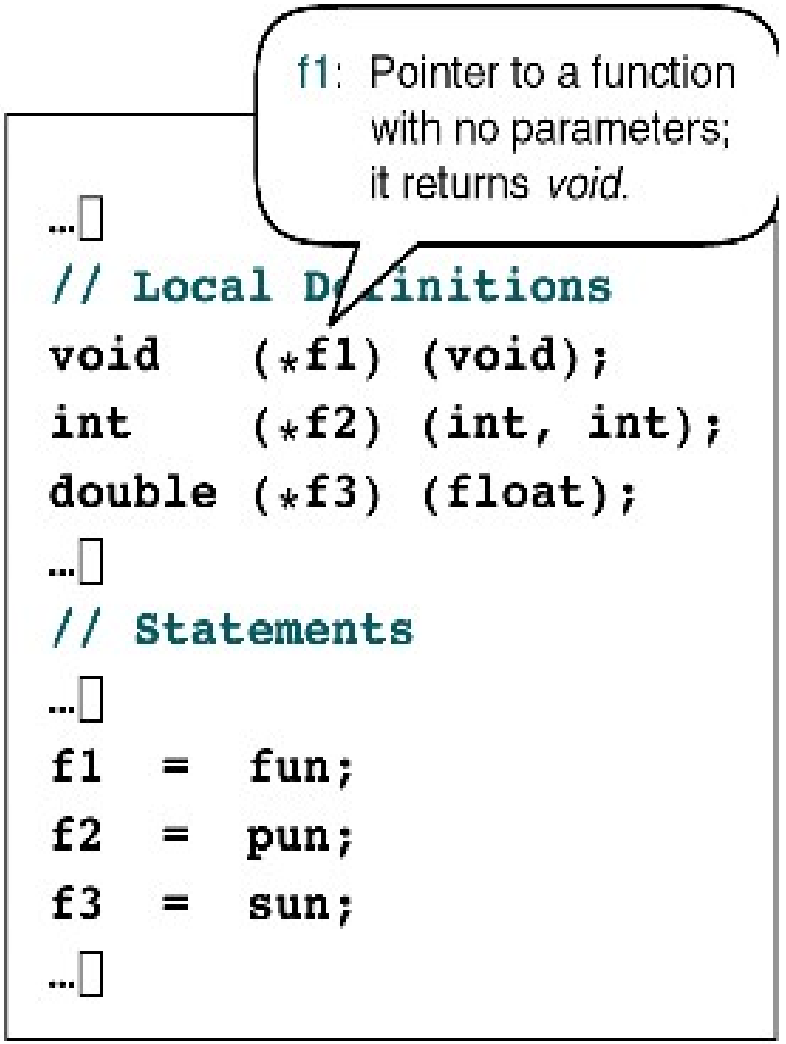


FIGURE 1-12 Pointers to Functions

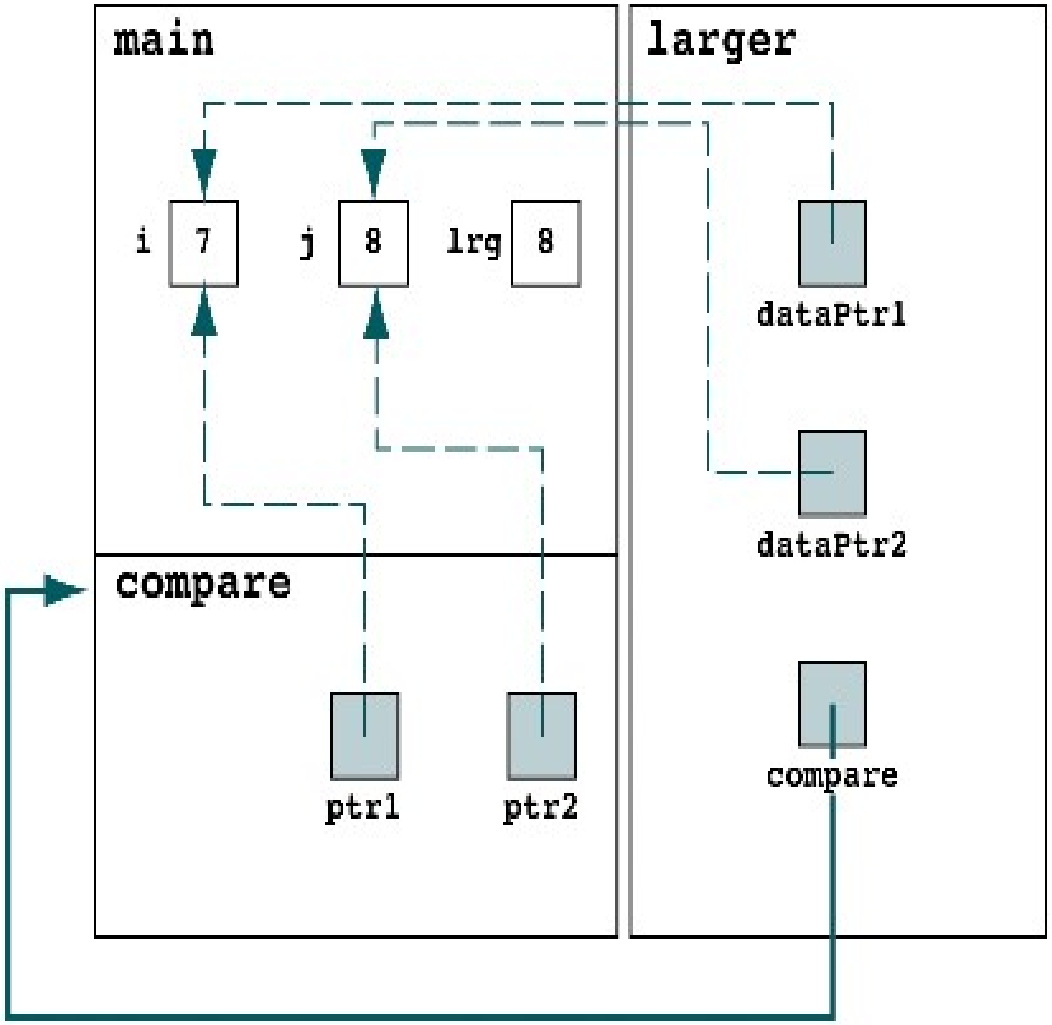


FIGURE 1-13 Design of Larger Function

```
void* larger (void* dataPtr1,    void* dataPtr2,
              int (*ptrToCF)(void*, void*))
{
   if ((*ptrToCF) (dataPtr1, dataPtr2) > 0)
       return dataPtr1;
   else
      return dataPtr2;
}       // larger
```

header file (Ch1A.h)

```
void* larger (void* dataPtr1,
          void* dataPtr2,
          int (*ptrToCF)(void*, void*))
{
   if
   ((*ptrToCF) (dataPtr1, dataPtr2) > 0)
       return dataPtr1;
   else
      return dataPtr2;
}            // larger
```

Label	Address	Value
	399	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
compare_I	5004 - 5125	102552
	…	

```
#include <stdio.h>
#include <stdlib.h>
#include "Ch1A.h" // Header file

int   compare (void* ptr1, void* ptr2);

int main (void)
{
   int i = 7 ;
   int j = 8 ;
   int lrg;
   lrg = (*(int*) larger (&i, &j, compare_I));

   printf ("Larger value is: %d\n", lrg);
   return 0;
}        // main

int compare_I (void* ptr1, void* ptr2)
{
  if (*(int*)ptr1 >=  *(int*)ptr2)
     return 1;
  else
     return -1;
}        // compare
```

header file (Ch1A.h)

```
void* larger (void* dataPtr1,
              void* dataPtr2,
              int (*ptrToCF)(void*, void*))
{
   if
   ((*ptrToCF) (dataPtr1, dataPtr2) > 0)
       return dataPtr1;
   else
      return dataPtr2;
}             // larger
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	
	…	
	…	
	…	
	…	
	…	
compare_I	5004 - 5125	102552
	…	

```
#include <stdio.h>
#include <stdlib.h>
#include "Ch1A.h" // Header file

int   compare (void* ptr1, void* ptr2);

int main (void)
{
   int i = 7 ;
   int j = 8 ;
   int lrg;
    lrg = (*(int*) larger (&i, &j, compare_I));

    printf ("Larger value is: %d\n", lrg);
    return 0;
}        // main

int compare_I (void* ptr1, void* ptr2)
{
  if (*(int*)ptr1 >=  *(int*)ptr2)
     return 1;
  else
     return -1;
}        // compare
```

header file (Ch1A.h)

```
void* larger (void* dataPtr1,
              void* dataPtr2,
              int (*ptrToCF)(void*, void*))
{
   if
   ((*ptrToCF) (dataPtr1, dataPtr2) > 0)
       return dataPtr1;
   else
      return dataPtr2;
}              // larger
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	
	…	
	…	
	…	
	…	
	…	
compare_I	5004 - 5125	102552
	…	

```
#include <stdio.h>
#include <stdlib.h>
#include "Ch1A.h" // Header file

int   compare (void* ptr1, void* ptr2);

int main (void)
{
   int i = 7 ;
   int j = 8 ;
   int lrg;
   lrg = (*(int*) larger (&i, &j, compare_I));

   printf ("Larger value is: %d\n", lrg);
   return 0;
}        // main

int compare_I (void* ptr1, void* ptr2)
{
  if (*(int*)ptr1 >=  *(int*)ptr2)
     return 1;
  else
     return -1;
}        // compare
```

header file (Ch1A.h)

```
void* larger (void* dataPtr1,
              void* dataPtr2,
              int (*ptrToCF)(void*, void*))
{
   if
   ((*ptrToCF) (dataPtr1, dataPtr2) > 0)
       return dataPtr1;
   else
      return dataPtr2;
}             // larger
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	
	…	
	…	
	…	
	…	
	…	
compare_I	5004 - 5125	102552
	…	

```
#include <stdio.h>
#include <stdlib.h>
#include "Ch1A.h" // Header file

int   compare (void* ptr1, void* ptr2);

int main (void)
{
   int i = 7 ;
   int j = 8 ;
   int lrg;
   lrg = (*(int*) larger (&i, &j, compare_I));

   printf ("Larger value is: %d\n", lrg);
   return 0;
}        // main

int compare_I (void* ptr1, void* ptr2)
{
  if (*(int*)ptr1 >=  *(int*)ptr2)
     return 1;
  else
     return -1;
}        // compare
```

```
void* larger (void* dataPtr1,
    void* dataPtr2,
    int (*ptrToCF)(void*, void*))
{
   if
   ((*ptrToCF) (dataPtr1, dataPtr2)
     > 0)
       return dataPtr1;
   else
      return dataPtr2;
}       // larger
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	
dataPtr1	484 - 487	400
dataPtr2	488 - 491	404
ptrToCF	492 - 495	5004
	...	
	...	
compare_I	5004 - 5125	102552
	...	

```
void* larger (void* dataPtr1,
    void* dataPtr2,
    int (*ptrToCF)(void*, void*))
{
   if
   ((*ptrToCF) (dataPtr1, dataPtr2)
     > 0)
       return dataPtr1;
   else
      return dataPtr2;
}       // larger
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	
dataPtr1	484 - 487	400
dataPtr2	488 - 491	404
ptrToCF	492 - 495	5004
	…	
	…	
compare_I	5004 - 5125	102552
	…	

```
void* larger (void* dataPtr1,
     void* dataPtr2,
     int (*ptrToCF)(void*, void*))
{
   if
   ((*ptrToCF) (dataPtr1, dataPtr2)
     > 0)
       return dataPtr1;
   else
      return dataPtr2;
}       // larger
```

```
int compare_I (void* ptr1, void* ptr2)
{
  if (*(int*)ptr1 >=  *(int*)ptr2)
     return 1;
  else
     return -1;
}       // compare
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	
dataPtr1	484 - 487	400
dataPtr2	488 - 491	404
ptrToCF	492 - 495	5004
ptr1	624 -627	400
ptr2	628 - 631	404
compare_I	5004 - 5125	102552
	...	

```
void* larger (void* dataPtr1,
     void* dataPtr2,
     int (*ptrToCF)(void*, void*))
{
    if
    ((*ptrToCF) (dataPtr1, dataPtr2)
      > 0)
        return dataPtr1;
    else
       return dataPtr2;
}       // larger
```

```
int compare_I (void* ptr1, void* ptr2)
{
  if (*(int*)ptr1 >=  *(int*)ptr2)
      return 1;
  else
      return -1;
}      // compare
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	
dataPtr1	484 - 487	400
dataPtr2	488 - 491	404
ptrToCF	492 - 495	5004
ptr1	624 -627	400
ptr2	628 - 631	404
compare_I	5004 - 5125	102552
	...	

if (7 >= 8)

```
void* larger (void* dataPtr1,
     void* dataPtr2,
     int (*ptrToCF)(void*, void*))
{
   if
   ((*ptrToCF) (dataPtr1, dataPtr2)
     > 0)
       return dataPtr1;
   else
      return dataPtr2;
}       // larger
```

```
int compare_I (void* ptr1, void* ptr2)
{
  if (*(int*)ptr1 >=  *(int*)ptr2)
     return 1;
  else
     return -1;
}       // compare
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	
dataPtr1	484 - 487	400
dataPtr2	488 - 491	404
ptrToCF	492 - 495	5004
ptr1	624 -627	400
ptr2	628 - 631	404
compare_I	5004 - 5125	102552
	...	

if (7 >= 8)

```
void* larger (void* dataPtr1,
     void* dataPtr2,
     int (*ptrToCF)(void*, void*))
{
    if
    ((*ptrToCF) (dataPtr1, dataPtr2)
      > 0)
        return dataPtr1;
    else
       return dataPtr2;
}       // larger
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	
dataPtr1	484 - 487	400
dataPtr2	488 - 491	404
ptrToCF	492 - 495	5004
	…	
	…	
compare_I	5004 - 5125	102552
	…	

if (-1 > 0)

```
void* larger (void* dataPtr1,
     void* dataPtr2,
     int (*ptrToCF)(void*, void*))
{
    if
    ((*ptrToCF) (dataPtr1, dataPtr2)
      > 0)
        return dataPtr1;
    else
        return dataPtr2;
}       // larger
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	
dataPtr1	484 - 487	400
dataPtr2	488 - 491	404
ptrToCF	492 - 495	5004
	…	
	…	
compare_I	5004 - 5125	102552
	…	

if (-1 > 0)

header file (Ch1A.h)

```
void* larger (void* dataPtr1,
              void* dataPtr2,
              int (*ptrToCF)(void*, void*))
{
   if
   ((*ptrToCF) (dataPtr1, dataPtr2) > 0)
       return dataPtr1;
   else
      return dataPtr2;
}             // larger
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	8
	…	
	…	
	…	
	…	
	…	
compare_I	5004 - 5125	102552
	…	

```
#include <stdio.h>
#include <stdlib.h>
#include "Ch1A.h" // Header file

int   compare (void* ptr1, void* ptr2);

int main (void)
{
   int i = 7 ;
   int j = 8 ;
   int lrg;
   lrg = (*(int*) larger (&i, &j, compare_I));

   printf ("Larger value is: %d\n", lrg);
   return 0;
}        // main

int compare_I (void* ptr1, void* ptr2)
{
  if (*(int*)ptr1 >=  *(int*)ptr2)
     return 1;
  else
     return -1;
}        // compare
```

lrg = *(int*) *address 404*

header file (Ch1A.h)

```
void* larger (void* dataPtr1,
              void* dataPtr2,
              int (*ptrToCF)(void*, void*))
{
   if
   ((*ptrToCF) (dataPtr1, dataPtr2) > 0)
       return dataPtr1;
   else
      return dataPtr2;
}             // larger
```

Label	Address	Value
	399	
i	400 - 403	7
j	404 - 407	8
lrg	408 - 411	8
	…	
	…	
	…	
	…	
	…	
compare_I	5004 - 5125	102552
	…	

```
#include <stdio.h>
#include <stdlib.h>
#include "Ch1A.h" // Header file

int   compare (void* ptr1, void* ptr2);

int main (void)
{
   int i = 7 ;
   int j = 8 ;
   int lrg;
   lrg = (*(int*) larger (&i, &j, compare_I));

   printf ("Larger value is: %d\n", lrg);
   return 0;
}        // main

int compare_I (void* ptr1, void* ptr2)
{
  if (*(int*)ptr1 >=  *(int*)ptr2)
     return 1;
  else
     return -1;
}        // compare
```

Larger value is: 8

header file (Ch1A.h)

```
void* larger (void* dataPtr1,
              void* dataPtr2,
              int (*ptrToCF)(void*, void*))
{
   if
   ((*ptrToCF) (dataPtr1, dataPtr2) > 0)
       return dataPtr1;
   else
      return dataPtr2;
}             // larger
```

Label	Address	Value
	399	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	
	…	

```
#include <stdio.h>
#include <stdlib.h>
#include "Ch1A.h" // Header file

int   compare (void* ptr1, void* ptr2);

int main (void)
{
   float f1 = 73.4;
   float f2 = 81.7;
   float lrg;
   lrg = (*(float*) larger(&f1,&f2, compare_F));

   printf ("Larger value is: %d\n", lrg);
   return 0;
}        // main

int compare_F (void* ptr1, void* ptr2)
{
  if (*(float*)ptr1 >=  *(float*)ptr2)
     return 1;
  else
     return -1;
}        // compare
```

changes required to make this compare two floating point values (notice func. **larger** does NOT change.

- passing func. pointer makes **larger** generic

Pointers to Functions in C

END OF SECTION