/* Multipurpose sorting program using function pointers */
#include <stdio.h>
define SIZE 10

void bubble(int *, const int, int (*)(int, int));
int ascending(const int, const int);
int descending(const int, const int);

main()
{
 int a[SIZE] = {2, 6, 4, 8, 10, 12, 89, 68, 45, 37};
 int counter, order;

 printf("Enter 1 to sort in ascending order,\n");
 printf("Enter 2 to sort in descending order: ");
 scanf("%d", &order);

 printf("nData items in original order\n");
 for (counter = 0; counter <= SIZE - 1; counter++)
 printf("%4d", a[counter]);

 if (order == 1) {
 bubble(a, SIZE, ascending);
 printf("nData items in ascending order\n");
 } else {
 bubble(a, SIZE, descending);
 printf("nData items in descending order\n");
 }

 for (counter = 0; counter <= SIZE - 1; counter++)
 printf("%4d", a[counter]);

 printf("\n");
 return 0;
}

void bubble(int *work, const int size, int (*)(int, int))
{
 int pass, count;
 void swap(int *, int *);

 for (pass = 1; pass <= size - 1; pass++)
 for (count = 0; count <= size - 2; count++)
 if (*compare)(work[count], work[count + 1])
 swap(&work[count], &work[count + 1]);

 void swap(int *element1Ptr, int *element2Ptr)
 {
 int temp;

temp = *element1Ptr;
*element1Ptr = *element2Ptr;
*element2Ptr = temp;

 int ascending(const int a, const int b)
 {
 return b < a;
 }

 int descending(const int a, const int b)
 {
 return b > a;
 }
PROGRAM example

VAR a, b, x: INTEGER;

PROCEDURE sub1;
VAR x, y: INTEGER;
BEGIN { sub1 }

VAR x: INTEGER;

PROCEDURE sub2;
BEGIN { sub2 }

PROCEDURE sub3;
VAR x: INTEGER;
BEGIN { sub3 }

BEGIN { Main Body of Program }

END.
Figure 4.1
The structure of a program

Figure 4.2
The tree structure of the program in Figure 4.1

Figure 4.3
The potential call graph of the program in Figure 4.1

Figure 4.4
The graph of the desirable calls in the program in Figure 4.1
Static Scope

```plaintext
program example;
  var a, b : integer;
...

procedure sub1;
  var x, y : integer;
  begin { sub1 }
  ...
  end; { sub1 }

begin { subl }
...
end; { subl }
```

Dynamic Scope

```plaintext
void sub1 (void) {
  int a, b;
  ...
} /* end of sub1 */

void sub2 (void) {
  int b, c;
  ...
  sub1;
} /* end of sub2 */

void main () {
  int c, d;
  ...
  sub2 ();
} /* end of main */
```

### Static Scope

- **Point**: Referencing Environment
  - 1: x and y of sub1, a and b of example
  - 2: x of sub3, (x of sub2 is hidden), a and b of example
  - 3: x of sub2, a and b of example
  - 4: a and b of example

### Dynamic Scope

- **Point**: Referencing Environment
  - 1: d of main, c of sub2, a and b of sub1 (c of main and b of sub2 are hidden)
  - 2: d of main, b of sub2 (c of main is hidden)
  - 3: c and d of main