1806ICT
Programming Fundamentals

C Functions

1

1

Topics

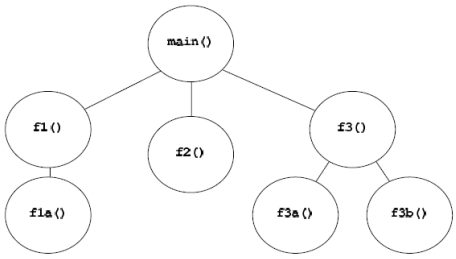
- Top-Down Design
- Functions
  - Parameters
  - Return values
  - Prototypes
  - Scope
  - Passing arrays as parameters
  - Passing character arrays as parameters

2

2

Top-Down Design

- Problem decomposition is at the heart of effective problem solving
- This is called the "top-down" method of programming
  - Start with a set of high-level tasks, and recursively splitting each task into subtasks until we get reasonably simple function modules



3

3

Topics

✓Top-Down Design

- Functions
  - Parameters
  - Return values
  - Prototypes
  - Scope

4

4

Functions

- A named sequence of instructions
- Also known as: modules, procedures, subroutines, ...
- Give a name to a standard, frequently used sequence of actions

5

5

Why use Functions?

- Simpler to correctly write a small function to do one job
- Debugging is easier
- Easier to maintain
- Small functions tend to be self-documenting and highly readable

6

6

Function Definition

Define a function as:

```
<functionname>
{
    <sequence of instructions>
}
```

7

7

Function Parameters

- Functions may have parameters
- They specify variations from call to call
- So that the same function can do different things
  - Depending on the value of the parameters

8

8

Function Parameters - continued

- For example:
  - √4 = 2
  - √36 = 6
- Both the above can be thought of as "calls" to the square root function
- But one returns 2, the other returns 6
  - Depending on the value of the parameter

9

9

Function Definition

Define a function with parameters as:

```
<functionname> ( <parameter>,
 <parameter>, ... )
{
     <sequence of instructions>
}
```

10

10

Writing User-defined Functions

- Create your own functions, similar to printf() or scanf()
- Need to specify:
  - the name of the function
  - its parameters
  - what it returns
  - block of statements to be carried out when the function is called
- The block of statements is called the "function body"

11

11

Example: hello1.c

```
Prints a simple greeting.


procedure sayHello
{
  output "Hello World!"
}


Main Program
{

  call procedure sayHello


}
```

12

12

Example: hello1.c

Prints a simple greeting.

```
procedure sayHello
{
  output "Hello World!"
}

Main Program
{
  do procedure sayHello
}
```

```
#include <stdio.h>
/*
 * Print a simple greeting.
 */
void sayHello ( void )
{
  printf("Hello World!\n");
}
/*
 * Call a function which
 * prints a simple greeting.
 */
int main(void)
{
  sayHello();
  return 0;
}
```

13

13

Example: hello1.c

Function definition

Function call

```
#include <stdio.h>
/*
 * Print a simple greeting.
 */
void sayHello ( void )
{
  printf("Hello World!\n");
}
/*
 * Call a function which
 * prints a simple greeting.
 */
int main(void)
{
  sayHello();
  return 0;
}
```

14

14

Example: hello1.c

Function name

Function body

```
#include <stdio.h>
/*
 * Print a simple greeting.
 */
void sayHello ( void )
{
  printf("Hello World!\n");
}
/*
 * Call a function which
 * prints a simple greeting.
 */
int main(void)
{
  sayHello();
  return 0;
}
```

15

15

Example: hello1.c

Return type

void = this function returns no value

Formal Parameter List

void = this function takes no arguments

```
#include <stdio.h>
/*
 * Print a simple greeting.
 */
void sayHello ( void )
{
  printf("Hello World!\n");
}
/*
 * Call a function which
 * prints a simple greeting.
 */
int main(void)
{
  sayHello();
  return 0;
}
```

16

16

Topics

- ✓Top-Down Design
- ✓Functions
  - Parameters
  - Return values
  - Prototypes
  - Scope

17

17

Parameters

- Information passed to a function
- "Formal" parameters are local variables declared in the function declaration.
- "Actual" parameters are values passed to the function when it is called.

18

18

Example: badsort.c

```
/* Print two numbers in order. */

void badSort ( int a, int b )
{
  if ( a > b )
  {
    printf("%d %d\n", b, a);
  }
  else
  {
    printf("%d %d\n", a, b);
  }
}
```

Parameters (aka Arguments)

19

19

Example: badsort.c

Formal parameters

```
/* Print two numbers in order.
   */

void badSort ( int a, int b )
{
  if ( a > b )
  {
    printf("%d %d\n", b, a);
  }
  else
  {
    printf("%d %d\n", a, b);
  }
}
```

Actual parameters

```
int main(void)
{
  int x = 3, y = 5;

  badSort ( 10, 9 );
  badSort ( y, x+4 );
  return 0;
}
```

20

20

Parameters (cont.)

- Parameters are passed by copying the value of the actual parameters to the formal parameters.
- Changes to formal parameters do not affect the value of the actual parameters.

21

21

Example: badswap.c

```
/* Swap the values of two
   variables. */
void badSwap ( int a, int b )
{
  int temp;

  temp = a;
  a = b;
  b = temp;
  printf("%d %d\n", a, b);
}
```

```
int main(void)
{
  int a = 3, b = 5;

  printf("%d %d\n",a,b);
  badSwap ( a, b );
  printf("%d %d\n",a,b);

  return 0;
}
```

22

22

Example: badswap.c

```
/* Swap the values of two
   variables. */
void badSwap ( int a, int b )
{
  int temp;
  temp = a;
  a = b;
  b = temp;
  printf("%d %d\n", a, b);
}
```

```
int main(void)
{
  int a = 3, b = 5;

  printf("%d %d\n",a,b);
  badSwap ( a, b );
  printf("%d %d\n",a,b);

  return 0;
}
```

Output:

```
3  5
```

23

Example: badswap.c

```
/* Swap the values of two
   variables. */
void badSwap ( int a, int b )
{
  int temp;
  temp = a;
  a = b;
  b = temp;
  printf("%d %d\n", a, b);
}
```

```
int main(void)
{
  int a = 3, b = 5;

  printf("%d %d\n",a,b);
  badSwap ( a, b );
  printf("%d %d\n",a,b);

  return 0;
}
```

Output:

```
3  5
5  3
```

24

Example: badswap.c

```
/* Swap the values of two
   variables. */
void badSwap ( int a, int b )
{
  int temp;
  temp = a;
  a = b;
  b = temp;
  printf("%d %d\n", a, b);
}
```

```
int main(void)
{
  int a = 3, b = 5;

  printf("%d %d\n",a,b);
  badSwap ( a, b );
  printf("%d %d\n",a,b);

  return 0;
}
```

Output:

```
3 5
5 3
3 5
```

25

Example: badswap.c

```
/* Swap the values of two
   variables. */
void badSwap ( int a, int b )
{
  int temp;
  temp = a;
  a = b;
  b = temp;
  printf("%d %d\n", a, b);
}
```

```
int main(void)
{
  int a = 3, b = 5;

  printf("%d %d\n",a,b);
  badSwap ( a, b );
  printf("%d %d\n",a,b);

  return 0;
}
```

Called function's environment:

```
a: 5
b: 3
```

Calling function's environment:

```
a: 3
b: 5
```

26

Parameters (cont.)

- Parameters are passed by copying the value of the actual parameters to the formal parameters.
- Changes to formal parameters do not affect the value of the actual parameters.

27

27

Parameters (cont.)

- If a function does not take parameters, declare its formal argument list void.

Declaration:

```
void sayHello ( void )
{
  printf("Hello World!\n");
}
```

Function call:

```
sayHello();
```

28

28

Topics

- ✓Top-Down Design
- ✓Functions
  - – Parameters
  - – Return values
  - – Prototypes
  - – Scope

29

29

Return Values

- Values are returned by copying a value specified after the **`return`** keyword

30

30

Example: max.c

Return type

```
/* Returns the larger of two
   numbers. */
int max (int a, int b)
{
  int result;

  if (a > b)
  {
    result = a;
  }
  else
  {
    result = b;
  }

  return result;
}
```

31

31

Example: max.c

```
/* Returns the larger of two
   numbers. */
int max (int a, int b)
{
  int result;

  if (a > b)
  {
    result = a;
  }
  else
  {
    result = b;
  }

  return result;
}
```

For example:

The value of the expression

max(7,5)

is the integer 7.

32

32

Return Values (cont.)

- If a function does not return a value, declare its return type void.

Declaration:

```
void sayHello ( void )
{
  printf("Hello World!\n");
}
```

Function call:

```
sayHello();
```

33

33

Topics

- ✓Top-Down Design
- ✓Functions
  - Parameters
  - Return values
  - Prototypes
  - Scope

34

34

Prototyping of Functions

- Must declare functions before use (like variables)
- Declaration is called a "prototype"
- Specifies the name, parameters and return type of the function, but not the code

35

Example: isNegative.c

```
#include <stdio.h>
int isNegative (int);
int main (void)
{
   int number;
   printf ("Enter an integer: ");
   scanf ("%d", &number);
   if (isNegative(number))
   {
       printf("Negative\n");
   }
   else
   {
       printf("Positive\n");
   }
   return 0;
}
```

```
int
isNegative ( int n )
{
   int result;
   if ( n<0 )
   {
       result = 1;
   }
   else
   {
       result = 0;
   }
   return result;
}
```

36

Example: isNegative.c

Function Prototype

```
#include <stdio.h>
int isNegative (int);
int main (void)
{
   int number;
   printf ("Enter an integer: ");
   scanf ("%d",&number);
   if (isNegative(number))
   {
       printf("Negative\n");
   }
   else
   {
       printf("Positive\n");
   }
   return 0;
}
```

```
int
isNegative ( int n )
{
   int result;
   if ( n<0 )
   {
       result=1;
   }
   else
   {
       result = 0;
   }
   return result;
}
```

37

Example: isNegative.c

```
#include <stdio.h>
int isNegative (int);
int main (void)
{
   int number;
   printf ("Enter an integer: ");
   scanf ("%d",&number);
   if (isNegative(number))
   {
       printf("Negative\n");
   }
   else
   {
       printf("Positive\n");
   }
   return 0;
}
```

Function Definition

```
int
isNegative ( int n )
{
   int result;
   if ( n<0 )
   {
       result=1;
   }
   else
   {
       result = 0;
   }
   return result;
}
```

38

Example: isNegative.c

```
#include <stdio.h>
int isNegative (int);
int main (void)
{
   int number;
   printf ("Enter an integer: ");
   scanf ("%d",&number);
   if (isNegative(number))
   {
       printf("  gative\n");
   }
   else
   {

   }
   retu
}
```

```
int
isNegative ( int n )
{
   int result;
   if ( n<0 )
   {
       result=1;
   }
   else
   {
       result = 0;
   }
   return result;
}
```

Function Call

(Must be after prototype, but can be before definition)

39

Example: isNegative.c

```
#include <stdio.h>
int isNegative (int);
int main (void)
{
   int number;
   printf ("Enter an integer: ");
   scanf ("%d",&number);
   if (isNegative(number))
   {
       printf("Negative\n");
   }
   else
   {
       printf("Positive\n");
   }
   return 0;
}
```

```
)
   if ( n<0 )
   {
       result=1;
   }
   else
   {
       result = 0;
   }
   return result;
}
```

Header files (filename.h) contain function prototypes and global variable declarations

40

Example: isNegative.c

```
#include <stdio.h>
int isNegative (int);
int main (void)
{
   int number;
   printf ("Enter an integer: ");
   scanf ("%d",&number);
   if (isNegative(number))
   {
       printf("Negative\n");
   }
   else
   {
       printf("Positive\n");
   }
   return 0;
}
```

```
int n )
{
    if ( n<0 )
    {
        result=1;
    }
    else
    {
        result = 0;
    }
    return result;
}
```

stdio.h contains function prototypes for `printf()`, `scanf()`, and other I/O functions

41

Topics

- ✓Top-Down Design
- ✓Functions
  - Parameters
  - Return values
  - Prototypes
  - Scope

42

42

Scope

Where can you use a variable which is declared in a function?

- In that function **only**

43

Scope: Local Variables

- Formal parameters: only accessible whilst function executing
- Variables declared in a function body: only accessible whilst function executing
- In fact, this is true of every block in a program

44

Example: isNegative.c

```
#include <stdio.h>

int
isNegative ( int n )
{
   int result;
   if (number<0)
   {
       result=1;
   }
   else
   {
       result = 0;
   }
   return result;
}
```

```
int main (void)
{
   int number;

   printf ("Enter an integer: ");
   scanf ("%d", &number);

   if (isNegative(number))
   {
       printf("Negative\n");
   }
   else
   {
       printf("Positive\n");
   }
   return 0;
}
```

45

Example: isNegative.c

```
#include <stdio.h>

int
isNegative ( int n )
{
   int result;
   if (number<0)
   {
       res   t=1;
   }
   else
   {
                = 0;
```

```
int main (void)
{
   int number;

   printf ("Enter an integer: ");
   scanf ("%d", &number);

   if (isNegative(number))
   {
       printf("Negative\n");
   }
   else
   {
       printf("Positive\n");

   rn 0;
```

ERROR! Number is local to the `main` function, not accessible here

46

Example: isNegative.c

```
#include <stdio.h>

int
isNegative ( int n )
{
   int result;
   if ( n<0 )
   {
       result=1;
   }
   else
   {
       result =
   }
   return resu
}
```

```
int main (void)
{
   int number;
   printf ("Enter an integer: ");
   scanf ("%d", &number);
   if (isNegative(number))
   {
       printf("Negative\n");
   }
   else

       ositive\n");

   return 0;
}
```

Use the parameter `n` which is local to the function `isNegative()`

47

Example: isNegative.c

```
#include <stdio.h>

int
isNegative ( int n )
{
   int result;
   if ( n<0 )
   {
       result=1;
   }
   else
   {
       result = 0;
   }
   return re
}
```

```
int main (void)
{
   int number;
   printf ("Enter an integer: ");
   scanf ("%d",&number);
   if (isNegative(number))
   {
       printf("Negative\n");
   }
   else
   {
       printf("Positive\n");
   }
```

`result` & `n`: local to `isNegative()`
`number`: local to `main()`

48

Scope: Global Variables

- Global variables are defined outside a function, usually on top of the program
- Global variables hold their values throughout the lifetime of the program
- They can be accessed inside any of the functions defined in the program.

49

Example: isNegativeGlobal.c

```
#include <stdio.h>
int number;
int
isNegative ( void )
{
   int result;
   if ( number <0 )
   {
       result=1;
   }
   else
   {
       result = 0;
   }
   return result;
}
```

```
int main (void)
{
   printf ("Enter an integer: ");
   scanf ("%d", &number);

   if (isNegative())
   {
       printf("Negative\n");
   }
   else
   {
       printf("Positive\n");
   }

   return 0;
}
```

50

Example: isNegativeGlobal.c

```
#include <stdio.h>
int number;
int
isNegative ( void )
{
   int result;
   if ( number <0 )
   {
       result=1;
   }
   else
   {
       result = 0;
   }
   return result;
}
```

```
int main (void)
{

   eger: ");

   ");

   else
   {
       printf("Positive\n");
   }

   return 0;
}
```

number is now GLOBAL - declared outside any function, accessible in all functions (after the declaration)

51

Scope: Global Variables

- Global variables are accessible in any function **after** their declaration to the end of that source file
- They're useful, but risky
  - if any and every function can modify them, it can be difficult to keep track of their value
- Better to use local variables and parameter passing if possible

52

Example: globalVariable.c

```
#include <stdio.h>
int number;
int addOne(void)
{
   number += 1;
   return number;
}
int addTwo(void)
{
   number += 2;
   return number;
}
int minusThree(void)
{
   number -= 3;
   return number;
}
```

```
int main (void)
{
   printf ("Enter an integer: ");
   scanf ("%d", &number);

   addOne();
   addTwo();
   minusThree();

   return 0;
}
```

53

static Storage Class Specifier

- Allows a local variable to retain its previous value when the function is re-entered/called again

54

Example: nonStaticVariable.c

```
#include <stdio.h>

void keepCount(void)
{
   int count = 0;

   printf("%d\n", count);

   count++;
}
```

```
int main (void)
{
   keepCount();
   keepCount();
   keepCount();
   return 0;
}
```

Output:

```
0
0
0
```

55

Example: staticVariable.c

```
#include <stdio.h>

void keepCount(void)
{
   static int count = 0;

   printf("%d\n", count);

   count++;
}
```

```
int main (void)
{
   keepCount();
   keepCount();
   keepCount();
   return 0;
}
```

count is now a static int

Output:

```
0
1
2
```

56

Scope: Functions

- Functions are also accessible in any function **after** their declaration to the end of that source file

57

Example: scopeFunctions.c

```
#include <stdio.h>
int area(int, int);
int peri(int, int);
void rectangle(int, int);

int area(int a, int b)
{
   return (a*b);
}
int peri(int a, int b)
{
   return 2*(a+b);
}
```

```
void rectangle(int a, int b)
{
   printf("Area = %d\n",
              area(a, b));
   printf("Perimeter = %d\n",
              peri(a, b));
}

int main (void)
{
   int side, len;
   scanf("%d %d", &side, &len);

   rectangle(side, len);
   return 0;
}
```

58

Passing Arrays to Functions

- The array is passed
  - as an array of unspecified size (`int array[]`)
- Changes to the array within the function affect the "original" array

59

59

Example 1: IORainfall-1

```
#include <stdio.h>
#define  NMONTHS 12

void loadRain ( int arrayPtr[] );
void printRain ( const int arrayPtr[] );

/* Store and print rainfall */
int main()
{
  int data[NMONTHS];

  loadRain(data);
  printRain(data);

  return 0;
}
```

60

60

Example 1: IORainfall-2

```
void loadRain ( int arrayPtr[] )
{
  int month;

  for (month=0; month < NMONTHS; month++)
  {
     scanf("%d", &arrayPtr[month]);
  }
}
```

61

61

Example 1: IORainfall-3

```
void printRain ( const int arrayPtr[] )
{
  int month;

  for (month=0; month < NMONTHS; month++)
  {
     printf("%5d ", arrayPtr[month]);
  }

  printf("\n");
}
```

62

62

Example 1: IORainfall -- (cont)

```
#include <stdio.h>
#define  NMONTHS 12
void loadRain ( int arrayPtr[] );
void printRain ( const int arrayPtr[] );
/* Store and print rainfall */
int main()
{
  int data[NMONTHS];
  loadRain(data);
  printRain(data);
  return 0;
}
/* Read in rainfall for each month*/
void loadRain ( int arrayPtr[] )
{
  int month;
  for (month=0; month < NMONTHS; month++)
  {  scanf("%d", &arrayPtr[month]);  }
}
/* Print rainfall for each month*/
void printRain ( const int arrayPtr[] )
{
  int month;
  for (month=0; month < NMONTHS; month++)
  {  printf("%5d", arrayPtr[month]);  }
  printf("\n");
}
```

63

63

Example 2: MinValue-1

```
#include <stdio.h>

int minimum ( int values[], int numElements );

/* Find the minimum value in an array */
int main()
{
  int array1[5] = {157, 324, 32, -12, 10};
  int array2[7] = {12, 43, 654, 23, 1, 10, 98};

  printf("array1 minimum = %d\n", minimum(array1, 5));
  printf("array2 minimum = %d\n", minimum(array2, 7));

  return 0;
}
```

64

64

Example 2: MinValue-2

```
int minimum ( int values[], int numElements )
{
  int minValue, i;

  minValue = values[0];

  for (i=1; i < numElements; i++)
  {
     if (values[i] < minValue)
        minValue = values[i];
  }

  return minValue;
}
```

65

65

Passing Two-Dimensional Arrays to Functions

- In the function definition, the declaration of a multidimensional array must have all sizes specified, except the first.
- Any changes to array elements within the function affect the "original" array elements

66

66

Example 2: 2-D Array-1

Both declarations are OK

```
#include <stdio.h>

#define  NROWS    3
#define  NCOLS    5

void inputEntry(float table[][NCOLS]);
void printTable(float table[NROWS][NCOLS]);

int main()
{
   float table[NROWS][NCOLS] = {{0}};

   printTable(table);

   while (1)
   {
      inputEntry(table);
      printTable(table);
   }
   return 0;
}
```

67

67

Example 2 (cont): 2-D Array-2

```
/* Reads in a location in the table and the value of
one item to be put in the table */
void inputEntry ( float table[][NCOLS] )
{
   int row, column;
   printf("Enter row and column number: ");
   scanf("%d %d", &row, &column);

   if ((0 <= row && row < NROWS) &&
       (0 <= column && column < NCOLS))
   {
      printf("Enter value: ");
      scanf("%f", &table[row][column]);
   }
   else
   {
      printf("Invalid entry location. No change.\n");
   }
}
```

68

68

Example 2 (cont): 2-D Array-3

```
/* Prints the table page-by-page, and each page
row-by-row */
void printTable ( float table[NROWS][NCOLS] )
{
   int   row, column;
   for (row=0; row < NROWS; row++)
   {
      for (column=0; column < NCOLS; column++)
      {
         printf("%10.2f", table[row][column]);
      }
      printf("\n");
   }
}
```

69

69

Character Strings as Function Parameters

- Strings as formal parameters are declared as **`char*`** or **`char[]`**
  - *Examples:*

```
void Greet ( char* name )
void Greet ( char name[] )
```

- They point to the first element of the string (array of chars)
- Changes to the string inside the function affect the actual string

70

70

Example: hello3.c

```
#include <stdio.h>
#include <string.h>
#define  NAMELEN  50

/* Print a simple greeting to
   the user */

void Greet ( char name[] )
{
  strcat(name, "! How are ya?");
}
```

```
int main()
{
  char user[NAMELEN];

  printf("Who are you? ");
  scanf("%s", user);
  Greet(user);
  printf("%s\n", user);

  return 0;
}
```

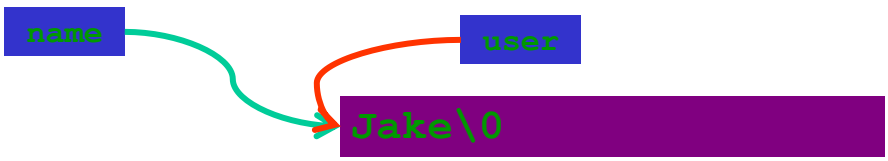
71

71

Example: hello3.c (cont)

```
#include <stdio.h>
#include <string.h>
#define  NAMELEN  50

/* Print a simple greeting to
   the user */

void Greet ( char * name )
{
  strcat(name, "! How are ya?");
}
```

```
int main()
{
  char user[NAMELEN];

  printf("Who are you? ");
  scanf("%s", user);
  Greet(user);
  printf("%s\n", user);

  return 0;
}
```



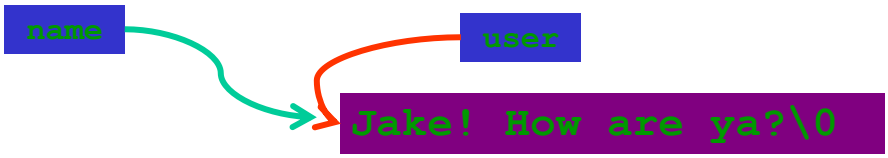
72

72

Example: hello3.c (cont)

```
#include <stdio.h>
#include <string.h>
#define  NAMELEN  50

/* Print a simple greeting to
   the user */

void Greet ( char * name )
{
  strcat(name, "! How are ya?");
}
```

```
int main()
{
  char user[NAMELEN];

  printf("Who are you? ");
  scanf("%s", user);
  Greet(user);
  printf("%s\n", user);

  return 0;
}
```



73

73

Example: hello3.c (cont)

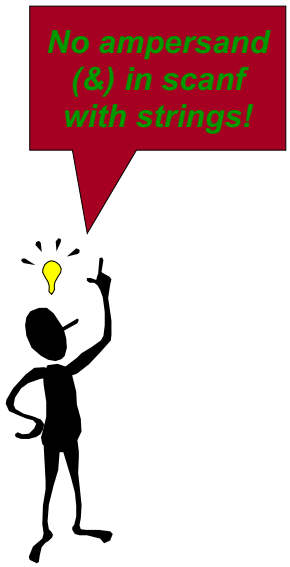
```
#include <stdio.h>
#include <string.h>
#define  NAMELEN  50

/* Print a simple greeting to
   the user */

void Greet ( char * name )
{
  strcat(name, "! How are ya?");
}
```

```
int main()
{
  char user[NAMELEN];

  printf("Who are you? ");
  scanf("%s", user);
  Greet(user);
  printf("%s\n", user);

  return 0;
}
```

74

74

More of *scanf* demystified




```
int main()
{
  char user[NAMELEN];

  printf("Who are you? ");
  scanf("%s", user);
  Greet(user);
  printf("%s\n", user);

  return 0;
}
```