

Running time of both versions

Worst case: the array is in reverse order

At pass i , the new element must move past all i preceding elements.

$$1 + 2 + \cdots + (n - 1) = \frac{n(n - 1)}{2}.$$

Both versions therefore take $\Theta(n^2)$ time in the worst case.

Best case: the array is already sorted

Each inner loop stops at its first check. There are no swaps or shifts, and the outer loop makes $n - 1$ passes.

Both versions take $\Theta(n)$ time on sorted input.

The improved version reduces copying while retaining the same asymptotic running time.

Insertion sort in **non-decreasing order** starts with:

9 4 7 2 6

What is the entire array after completing the pass $i = 2$?

Pass $i = 1$ happens first. Array indices start at 0.

Poll response: Type all five values, separated by spaces.

Discuss in your group and submit one answer.

1 mark

The improved version is about to insert $\text{key} = 5$ at pass $i = 3$.

2 6 9 5

```
while (j >= 0 && A[j] > key) {  
    A[j+1] = A[j];  
    j--;  
}  
A[j+1] = key;
```

How many shifts occur, and at which index is key inserted?

Poll response: Type the number of shifts and the insertion index.

Discuss in your group and submit one answer.

1 mark

Fill the condition so that the loop safely sorts in **non-decreasing order**.

```
int key = A[i];
int j = i - 1;
while ( /* condition */ ) {
    A[j+1] = A[j];
    j--;
}
A[j+1] = key;
```

The loop must never read a negative array index.

Poll response: Type the complete condition, including the bounds check.

Discuss in your group and submit one answer.

1 mark

Insertion sort: answers

I1: 4 7 9 2 6

Pass 1 inserts 4 before 9. Pass 2 inserts 7 between 4 and 9.

I2: 2 1

Shift 9 and 6 right. The loop ends with $j = 0$, so 5 goes into $A[1]$.

I3: $j \geq 0 \ \&\& \ A[j] > \text{key}$

C evaluates $\&\&$ left to right and skips the second operand when the first is false.

One mark each. For I2, award half a mark for each correct value.

Character grids

Find the rows and columns containing cse as consecutive characters. Search left to right in rows and top to bottom in columns.

	0	1	2	3	4
0	c	a	t	t	y
1	c	c	s	e	p
2	e	s	c	e	l
3	s	e	e	s	e
4	h	a	p	s	a

`find(text, pattern)` returns:

- the starting index of the first match, or
- -1 when there is no match.

A match at index 0 is valid.

Character grids

Find the rows and columns containing cse as consecutive characters. Search left to right in rows and top to bottom in columns.

	0	1	2	3	4
0	c	a	t	t	y
1	c	c	s	e	p
2	e	s	c	e	l
3	s	e	e	s	e
4	h	a	p	s	a

`find(text, pattern)` returns:

- the starting index of the first match, or
- -1 when there is no match.

A match at index 0 is valid.

Subtasks: Search each row. Then search each column.

Character-grid search: subtasks

The function `find` returns an index or `-1`.

```
for each row R of grid
    if find(R, s) >= 0
        print index of R
gridT = transpose(grid)
for each row R of gridT
    if find(R, s) >= 0
        print index of R
```

Character-grid search: subtasks

The function `find` returns an index or `-1`.

```
for each row R of grid
    if find(R, s) >= 0
        print index of R
gridT = transpose(grid)
for each row R of gridT
    if find(R, s) >= 0
        print index of R
```

Functions let us implement searching and transposing separately, then use them in the main program.

Each row supplied to a string function needs a terminating null character. Transposing turns each original column into a row.

Selection sort: subtasks

Task: sort an array in non-increasing order.

- `FindMax(i, n-1)` finds the index of the largest element from position `i` through `n-1`.
- `Swap(i, j)` exchanges the elements at two positions.

Selection sort: subtasks

Task: sort an array in non-increasing order.

- `FindMax(i, n-1)` finds the index of the largest element from position `i` through `n-1`.
- `Swap(i, j)` exchanges the elements at two positions.

```
for i = 0 to n-2
    maxindex = FindMax(i, n-1)
    Swap(i, maxindex)
```

Pseudocode: both subtasks operate on the same array. We will later specify how to give a function access to that array.

Defining our own functions

- `sqrt(x)` computes a square root.
- `pow(x, k)` computes x^k .

Defining our own functions

- `sqrt(x)` computes a square root.
- `pow(x, k)` computes x^k .
- We can define `gcd(m, n)` to compute a greatest common divisor.
- We can define `FindSum(a, b)` to add two integers.

Defining our own functions

- `sqrt(x)` computes a square root.
- `pow(x, k)` computes x^k .
- We can define `gcd(m, n)` to compute a greatest common divisor.
- We can define `FindSum(a, b)` to add two integers.

A **function** groups the instructions for a task under a name. A function call runs those instructions.

- Divide a larger task into smaller subtasks.

- Divide a larger task into smaller subtasks.
- Implement and test each subtask separately.

- Divide a larger task into smaller subtasks.
- Implement and test each subtask separately.
- Reuse the same function with different inputs.

This is a **modular approach** to program design.

A function needs a declaration

```
#include <stdio.h>
int main()
{
    int var1 = 10;
    int var2 = 20;
    int var3, var4;
    var3 = FindSum(var1, var2);
    var4 = FindSum(var3, var2);
    printf("%d", var3);
    printf("%d", var4);
    return 0;
}
```

What happens when this program is compiled?

A function needs a declaration

```
#include <stdio.h>
int main()
{
    int var1 = 10;
    int var2 = 20;
    int var3, var4;
    var3 = FindSum(var1, var2);
    var4 = FindSum(var3, var2);
    printf("%d", var3);
    printf("%d", var4);
    return 0;
}
```

What happens when this program is compiled?

- FindSum has no declaration before its calls.
- Modern C requires a declaration before use.
- We also need a definition to link a complete program.

Declaration and definition

```
#include "stdio.h"
int FindSum(int, int);
int main()
{
    int var1 = 10;
    int var2 = 20;
    int var3, var4;
    var3 = FindSum(var1, var2);
    var4 = FindSum(var3, var2);
    printf("%d\n", var3);
    printf("%d\n", var4);
    return 0;
}
int FindSum(int a, int b)
{
    int c=a+b;
    return c;
}
```

Prototype

- Function name.
- Return type.
- Number and types of parameters.

Declaration and definition

```
#include "stdio.h"
int FindSum(int, int);
int main()
{
    int var1 = 10;
    int var2 = 20;
    int var3, var4;
    var3 = FindSum(var1, var2);
    var4 = FindSum(var3, var2);
    printf("%d\n", var3);
    printf("%d\n", var4);
    return 0;
}
int FindSum(int a, int b)
{
    int c=a+b;
    return c;
}
```

Prototype

- Function name.
- Return type.
- Number and types of parameters.

Definition

Adds parameter names and a body with the instructions to execute.

return c; supplies the result to the caller.

Definition before use

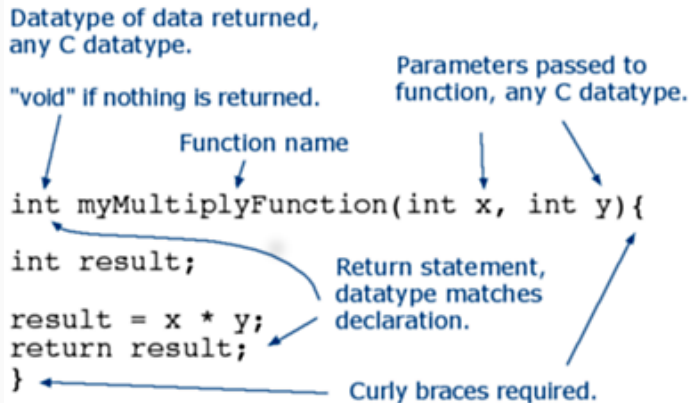
```
#include "stdio.h"
int FindSum(int a, int b)
{
    int c=a+b;
    return c;
}
int main()
{
    int var1 = 10;
    int var2 = 20;
    int var3,var4;
    var3 = FindSum(var1,var2);
    var4 = FindSum(var3,var2);
    printf("%d\n", var3);
    printf("%d\n", var4);
    return 0;
}
```

The definition appears before main calls FindSum.

It also declares the function, so a separate prototype is unnecessary here.

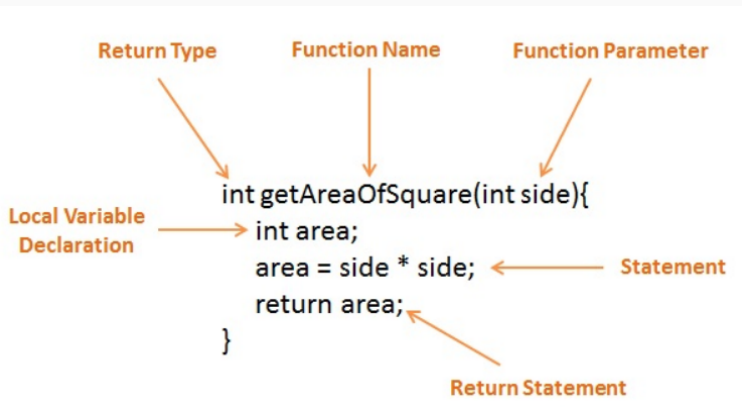
Anatomy of a function definition

myMultiplyFunction returns the product of two integers.



A function that computes an area

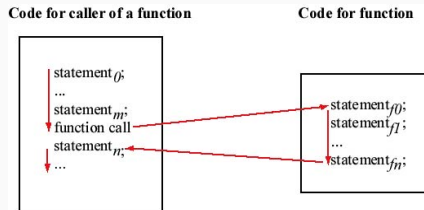
`getAreaOfSquare(side)` returns the area of a square. If `side` is in centimetres, the result is in cm^2 .



Function calls

```
var3 = FindSum(var1, var2);  
var4 = FindSum(var3, var2);
```

- The caller supplies argument values.
- The function executes and returns a result.
- Execution resumes at the caller.



A factorial function

```
#include<stdio.h>
int fact(int);
int main() {
    int x, y;
    printf("Enter a number:");
    scanf("%d", &x);
    y = fact(x);
    printf("%d\n",y);
}
int fact(int n) {
    int i = 1;
    while(n>0) {
        i = i * n;
        n--;
    }
    return i;
}
```

Assume $0 \leq x \leq 12$ and a 32-bit int.

- A function body can contain loops and local variables.
- `fact(0)` returns 1.
- `n` receives a copy of the argument value.

Library functions and headers

- `math.h` declares functions such as `sqrt` and `pow`.
- `stdio.h` declares functions such as `printf` and `scanf`.

The standard library supplies their implementations. Including a header makes the declarations available to the compiler.

Arguments and no return value

```
#include<stdio.h>
void area(float rad); // Prototype Declaration
int main()
{
    float rad;
    printf("Enter the radius : ");
    scanf("%f",&rad);
    area(rad);
}
void area(float rad)
{
    float ar;
    ar = 3.14 * rad * rad ;
    printf("Area of Circle = %f",ar);
}
```

area(rad) receives the radius and prints an area.

Return type void means that the function returns no value.

No arguments and no return value

```
#include<stdio.h>
void area(void); // Prototype Declaration
int main(void)
{
    area();
    return 0;
}
void area(void)
{
    float area_circle;
    float rad;
    printf("Enter the radius : ");
    scanf("%f",&rad);
    area_circle = 3.14 * rad * rad ;
    printf("Area of Circle = %f",area_circle);
}
```

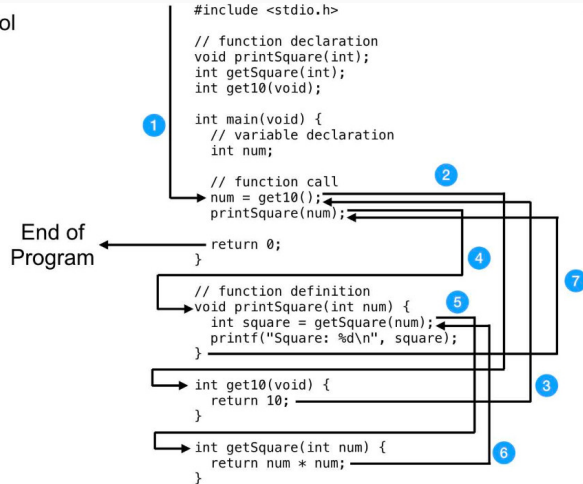
`void area(void)` takes no arguments and returns no value.

This version reads the radius inside the function.

`main` returns an `int`.

Control flow across function calls

Flow of control



Using functions

- Choose a subtask with a clear input and result.
- Write its declaration and definition.
- Call it wherever the program needs that subtask.
- Test it with several argument values.

What exactly is printed?

```
int FindSum(int a, int b) {  
    return a + b;  
}  
  
int main(void) {  
    int x = 4;  
    int y = FindSum(x, 3);  
    x = FindSum(y, x);  
    printf("%d %d", x, y);  
    return 0;  
}
```

Assume `#include <stdio.h>`.

Poll response: Type the two printed integers.

Discuss in your group and submit one answer.

1 mark

Write a prototype for a function named `larger` that:

- takes two `int` arguments, and
- returns an `int`.

Place the prototype before `main`.

Poll response: Type one complete declaration, including its semicolon.

Discuss in your group and submit one answer.

1 mark