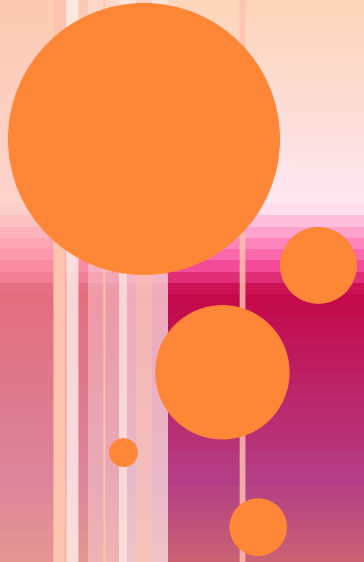
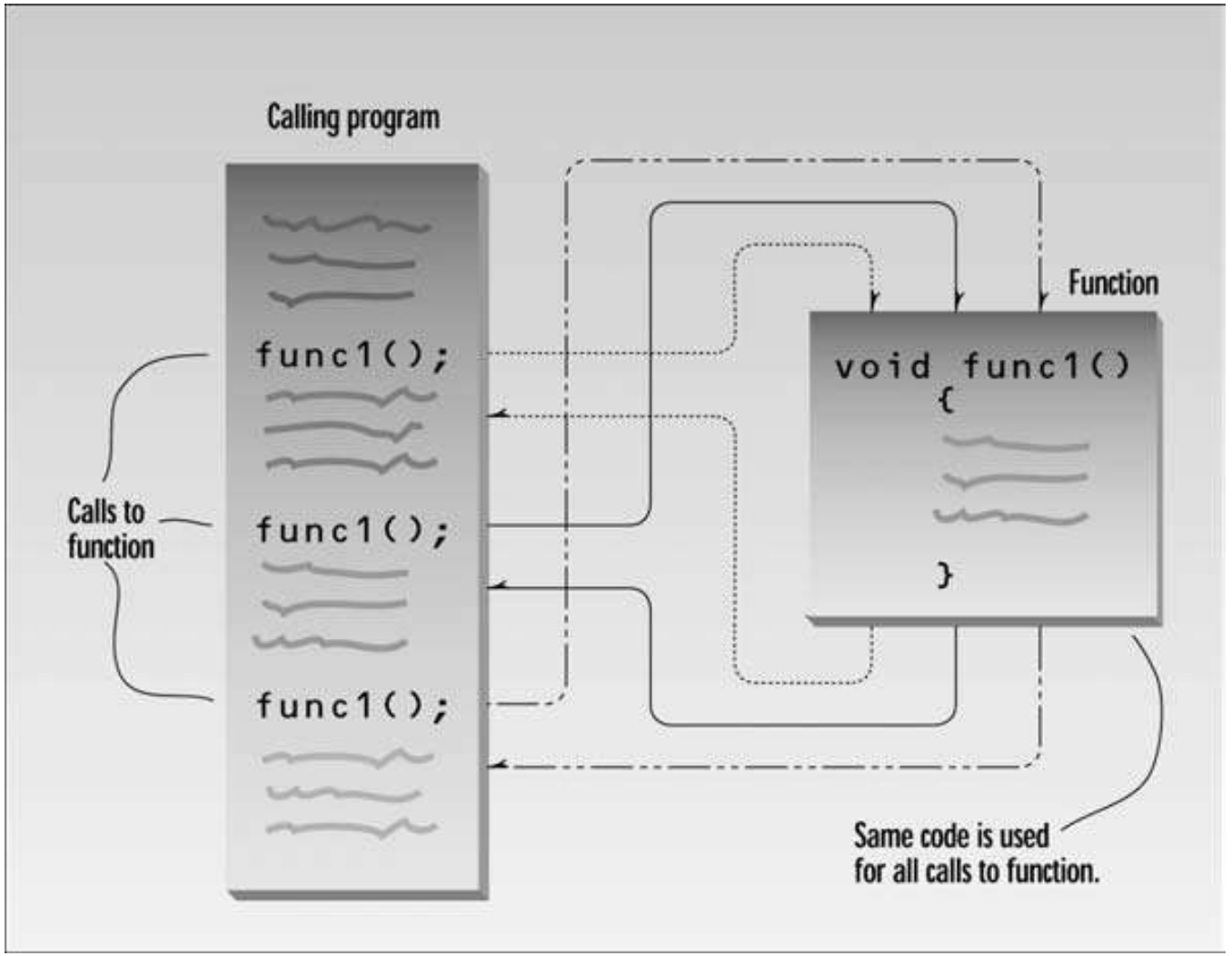
FUNCTIONS IN C++



Introduction to functions

- A function groups a number of program statements into a unit and gives it a name.
- This unit can then be invoked from other parts of the
- Another reason to use functions is to reduce program size.
- Any sequence of instructions that appears in a program more than once is a candidate for being made into a function. program.

Flow of control to a function



```
#include <iostream>
void starline();          //function declaration/ (prototype)
int main()
{
starline();               //call to function
cout << "Data type Range" << endl;
starline();               //call to function
cout << "char -128 to 127" << endl
starline();               //call to function
return 0;
}
// function definition
void starline()           //function declarator/definition
{
for(int j=0; j<45; j++) //function body
cout << '*';
}
```

The Function Declaration

- The declaration tells the compiler that at some later point we plan to present a function called *starline*.
- *The **keyword void** specifies that the function has no return value, and the empty* parentheses indicate that it takes no **arguments.**
- Function declarations are also called *prototypes*

Calling the Function

- The function is *called (or invoked, or executed) three times from main().*
- *Each of the three* calls looks like this:

starline();

- This is all we need to call the function:

1) the function name
2) followed by parentheses.

The syntax of the call is very similar to that of the declaration, **except that the return type is not used.**

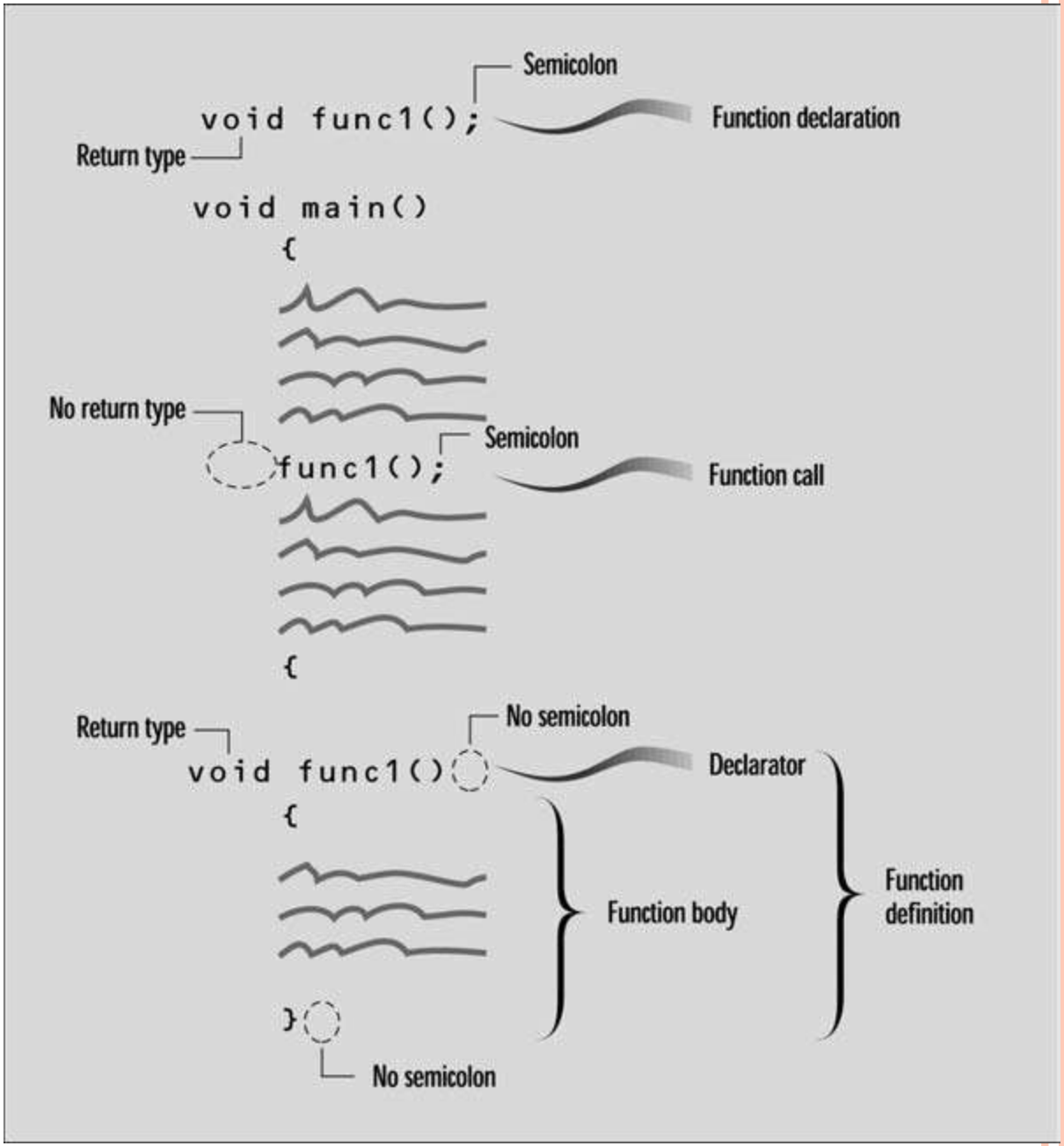
3)The call is terminated by a semicolon.

The Function Definition

```
void starline() //declarator
{
for(int j=0; j<45; j++) //function body
cout << '*';
cout << endl;
}
```

The definition consists of a line called the *declarator, followed by the function body. The* function body is composed of the statements that make up the function, delimited by braces.

FUNCTION SYNTAX



Function Components

Table 5.1 Function Components

Component	*Purpose*	*Example*
Declaration (prototype)	Specifies function name, argument types, and return value. Alerts compiler (and programmer) that a function is coming up later.	`void func();`
Call	Causes the function to be executed.	`func();`

Definition	The function itself. Contains the lines of code that constitute the function.	`void func()`
		`{`
		`// lines of code`
		`}`

Declarator	First line of definition.	`void func()`

Eliminating the Declaration

```
#include <iostream>
// starline() //function definition
void starline()
{
for(int j=0; j<45; j++)
cout << '*';
cout << endl;
}
int main()
{
starline(); //call to function
cout << "Data type Range" << endl;
starline(); //call to function
cout << "char -128 to 127"
starline();
```

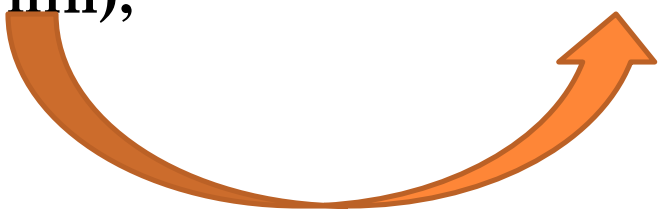
Passing Arguments to Functions

- An *argument is a piece of data (an int value, for example) passed from a program to the function.*
- Arguments allow a function to operate with different values, or even to do different
- things, depending on the requirements of the program calling it.

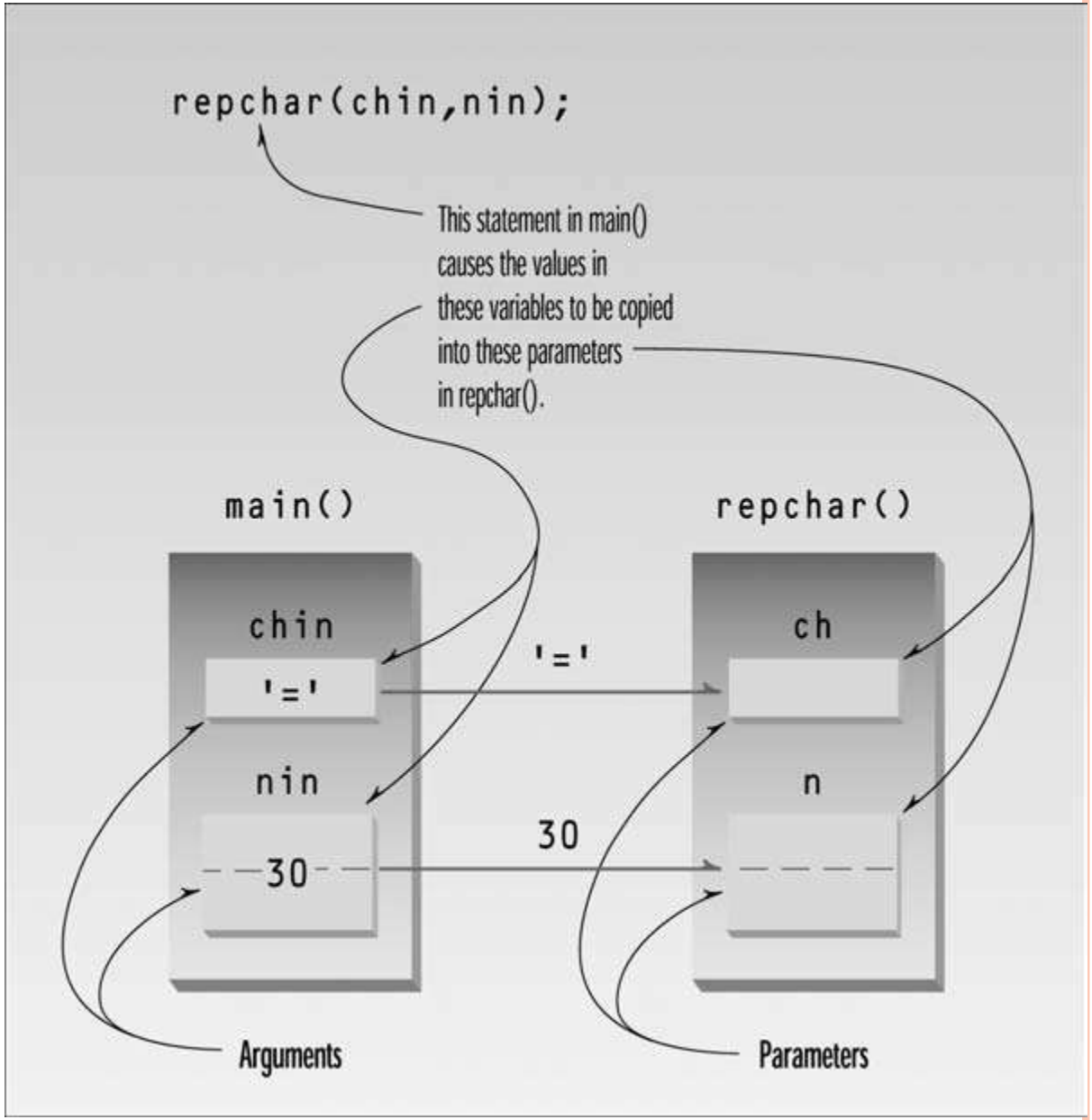
Passing Variables

```
#include <iostream>
void repchar(char, int); //function
   declaration
int main()
{
char chin;
int nin;
cout << "Enter a character: ";
cin >> chin;
cout << "Enter number of times
   to repeat it: ";
cin >> nin;
repchar(chin, nin);
return 0;
}
```

```
// function definition
void repchar(char ch, int
n) //function declarator
{
for(int j=0; j<n; j++)
//function body
cout << ch;
cout << endl;
}
```



Passing by value.



Returning Values from Functions

- When a function completes its execution, it can return a single value to the calling program.

Returning Values from Functions

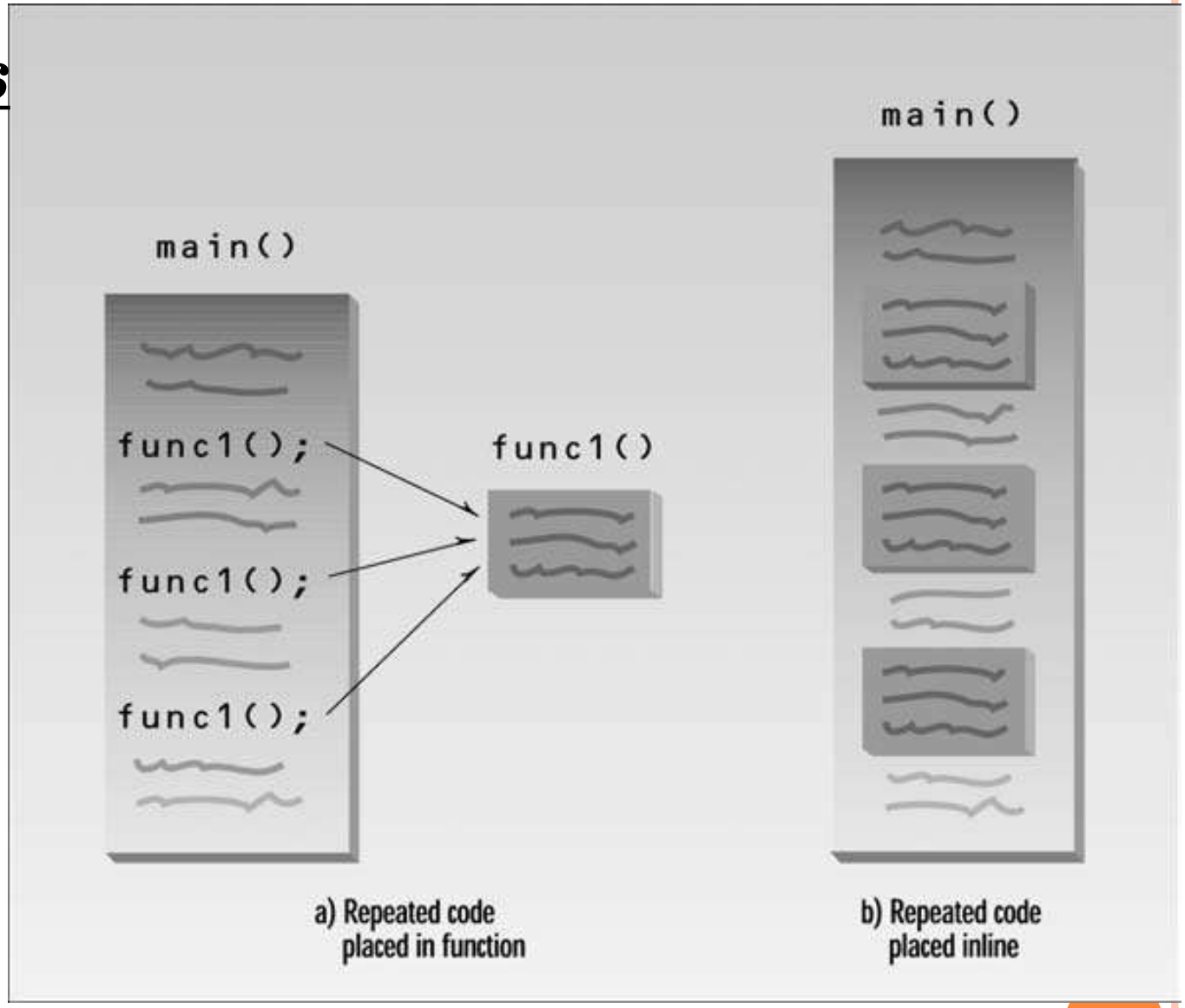
```
#include <iostream>
float lbstokg(float);
   //declaration
int main()
{
float lbs, kgs;
cout << "\nEnter your
   weight in pounds: ";
cin >> lbs;
kgs = lbstokg(lbs);
cout << "Your weight in
   kilograms is " << kgs <<
   endl;
return 0;
```

```
float lbstokg(float pounds)
{
float kilograms = 0.453592
* pounds;
return kilograms;
}
```

Inline Functions

To save execution time in short functions, you may elect to put the code in the function body directly inline with the code in the calling program. That is, each time there's a function call in the source file, **the actual code from the function is inserted, instead of a jump to the function**

Functions versus inline code



Inline function

This kind of function is written like a normal function in the source file **but compiles into inline code instead of into a function**.

The source file remains well organized and easy to read, since the function is shown as a separate entity.

However, when the program is compiled, the function body is actually inserted into the program wherever a function call occurs.

```
#include <iostream>
inline float lbstokg(float pounds)
{
return 0.453592 * pounds;
}
 int main()
{
float lbs;
cout << “\nEnter your weight in pounds: “;
cin >> lbs;
cout << “Your weight in kilograms is “ << lbstokg(lbs)
return 0;
}
```


Thank
You