

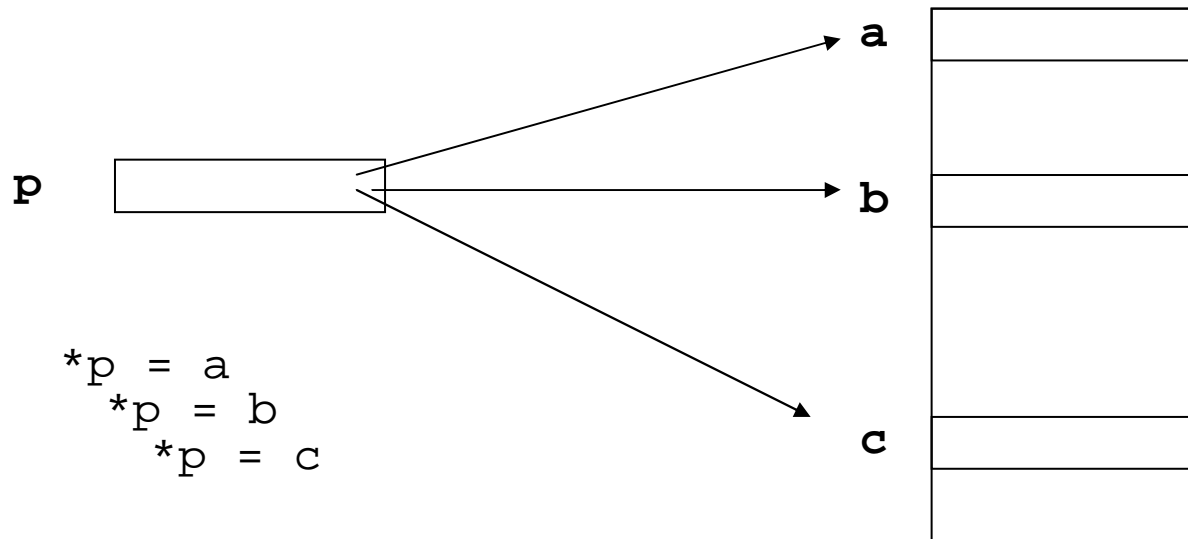
Chapter 6



More about Program Structures

Pointers

- A pointer stores an address
 - which point to a variable of some type
- A single pointer can point to different variables at different times



Pointers to Functions

- A pointer to functions also provide you the flexibility.
 - It will call the function whose address was last assigned to the pointer.
- A pointer to a function must contain
 - The memory address of the function
 - The parameter list
 - The return type

Declaring Pointers to Functions

- ❑ `double (*pfun) (char*, int);`
 - The parentheses around the pointer name, `pfun`, and the asterisk are necessary.
 - Otherwise, `double *pfun (char*, int)` would be a function returning a pointer to a double value.

- ❑ `long sum(long num1, long num2);`
- ❑ `long (*pfun)(long, long) = sum;`

- ❑ `long product(long, long);`
- ❑ `pfun = product;`

Ex6_01.cpp on P.273

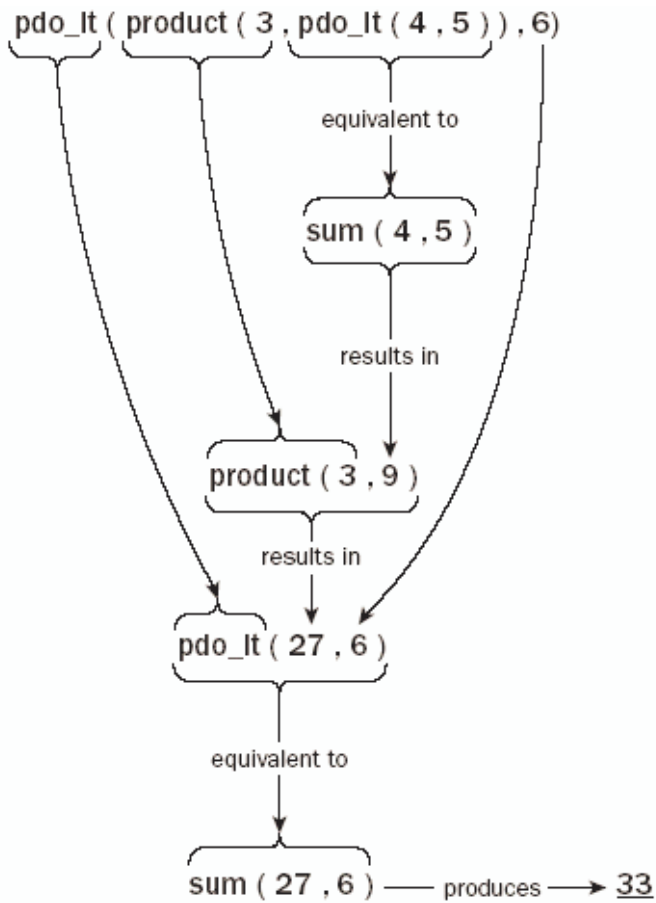


Figure 6-1

A Simpler Example

- As a matter of fact, I think Ex6_01.cpp is too complicated. I prefer the following example:

- `pdo_it = product;`
- `cout << pdo_it(3,5) << endl;`
- `pdo_it = sum;`
- `cout << pdo_it(3,5) << endl;`

A Pointer to a Function as an Argument

- Ex6_02.cpp on P.275

Arrays of Pointers to Functions

- ❑ `double sum(double, double);`
- ❑ `double product(double, double);`
- ❑ `double difference(double, double);`
- ❑ `double (*pfun[3]) (double, double) =
{ sum, product, difference } ;`
 - `pfun[1](2.5, 3.6);`
 - `(*pfun)(2.5, 3.6);`
 - `(* (pfun+1)) (2.5, 3.6);`

Initializing Function Parameters

- ❑ You may declare the default value of some parameters:
 - `void showit(char msg[] = "I know the default!");`
- ❑ When you omit the argument in calling the function, the default value will be supplied automatically.
 - `showit("Today is Wednesday.");`
 - `showit();`
- ❑ Notes on P.279: Only the last argument(s) can be omitted.

Function Overloading

- ❑ **Function overloading** allows you to use the same function name for defining several functions as long as they each have different parameter lists.
- ❑ When the function is called, the compiler chooses the correct version according to the list of arguments you supply.
- ❑ The following functions share a common name, but have a **different parameter list**:
 - `int max(int array[], int len);`
 - `long max(long array[], int len);`
 - `double max(double array[], int len);`

Ex6_07.cpp on P.287

- ❑ Three overloaded functions of max()
- ❑ In main(), C compiler inspect the argument list to choose different version of functions.

Signature

- ❑ The signature of a function is determined by its name and its parameter list.
- ❑ All functions in a program must have unique signatures
- ❑ The following example is not valid overloading
 - `double max(long array[], int len);`
 - `long max(long array[], int len);`
- ❑ A different return type does not distinguish a function, if the signatures are the same.

Function Templates

- ❑ In Ex6_07.cpp, you still have to repeat the same code for each function, with different variable and parameter types.
- ❑ You may define a **function template** to ask C compiler automatically generate functions with various parameter types.

Defining a Function Template

```
template<typename T> T max(T x[], int len)
{
    T max = x[0];
    for (int i = 1; i < len; i++)
        if (max < x[i])
            max = x[i];
    return max;
}
```

Using a Function Template

- ❑ Each time you use the function `max()` in your program, the compiler checks to see if a function corresponding to the type of arguments that you have used in the function call already exists.
 - If the function does not exist, the compiler creates one by substituting the argument type in your function call to replace the parameter `T`.
- ❑ Compare `Ex6_08.cpp` and `Ex6_07.cpp` to see how the source code is reduced.
 - Note that using a template doesn't reduce the size of your **compiled** program.
- ❑ Q: Can we calculate the length of the array **inside** the function?

宣州謝朓樓餞別校書叔雲

~ 李白

- 棄我去者 昨日之日不可留
- 亂我心者 今日之日多煩憂
- 長風萬里送秋雁 對此可以酣高樓
- 蓬萊文章建安骨 中間小謝又清發
- 俱懷逸興壯思飛 欲上青天覽明月
- 抽刀斷水水更流 舉杯消愁愁更愁
- 人生在世不稱意 明朝散髮弄扁舟

Topics in This Semester

Part 1: Object-Oriented Programming

- Defining Your Own Data Types
- More on Classes
- Class Inheritance and Virtual Functions

Part 2: Windows Programming

- Windows Programming Basics
- Microsoft Foundation Classes (MFC)
- Working with Menus and Toolbars
- Drawing in a Window
- Creating the Document and Improving the View
- Working with Dialogs and Controls
- Storing and Printing Documents
- Writing Your Own DLLs (optional)

Part 3: Database Application

- Connecting to Data Sources
- Updating Data Sources
- Applications Using Windows Forms
- Accessing Data Sources in a Windows Forms Application

Chapter 1: Programming with Visual C++ 2005

- The .NET Framework
- The Common Language Runtime (CLR)
- Using the Integrated Development Environment (IDE)

Chapter 2: Data, Variables, and Calculations

- Defining Variables
- Fundamental Data Types
 - Integer
 - Character
 - Boolean
 - Floating-Point
- Basic Input/Output Operations
- Variable Types and Casting
- Storage Duration and Scope

Chapter 3: Decisions and Loops

- Comparing Values
 - if ... else ...
 - switch
 - The Conditional Operator
- Repeating a Block of Statements
 - for loop
 - continue
 - break
 - while loop
 - do-while loop
- Nested loop

Chapter 4: Arrays, Strings, and Pointers

- Arrays
 - Declaring Arrays
 - Initializing Arrays
 - Character Arrays
 - Multidimensional Arrays
- Indirect Data Access – Pointer
- Dynamic Memory Allocation

Chapter 5: Functions

- ❑ Structure of a Function
- ❑ Passing Arguments to a Function
 - The Pass-by-value Mechanism
 - Pointers as Arguments to a Function
 - References as Arguments to a Function
- ❑ Returning Values from a Function
 - Returning a Pointer
 - Returning a Reference
 - Static Variables in a Function
- ❑ Recursive Function Calls

Chapter 6: More about Functions

- Pointers to Functions
- Initializing Function Parameters
- Exceptions
- Function Overloading
- Function Templates

Case Study:

Implementing a Calculator

□ Goal

- Design a program which acts as a calculator.
- It will take an arithmetic expression, evaluate it, and print out the result.
- For example, taking the input string
“2 * 3.14159 * 12.6 * 12.6 /2 + 25.2 * 25.2”
will obtain the result “1133.0”.

□ To make it simple at the first stage,

- The whole computation must be entered in a single line.
- Spaces are allowed to be placed anywhere.
- Parentheses are not allowed in the expression.
- Only unsigned numbers are recognized.

Step 1: Eliminating Blanks from a String

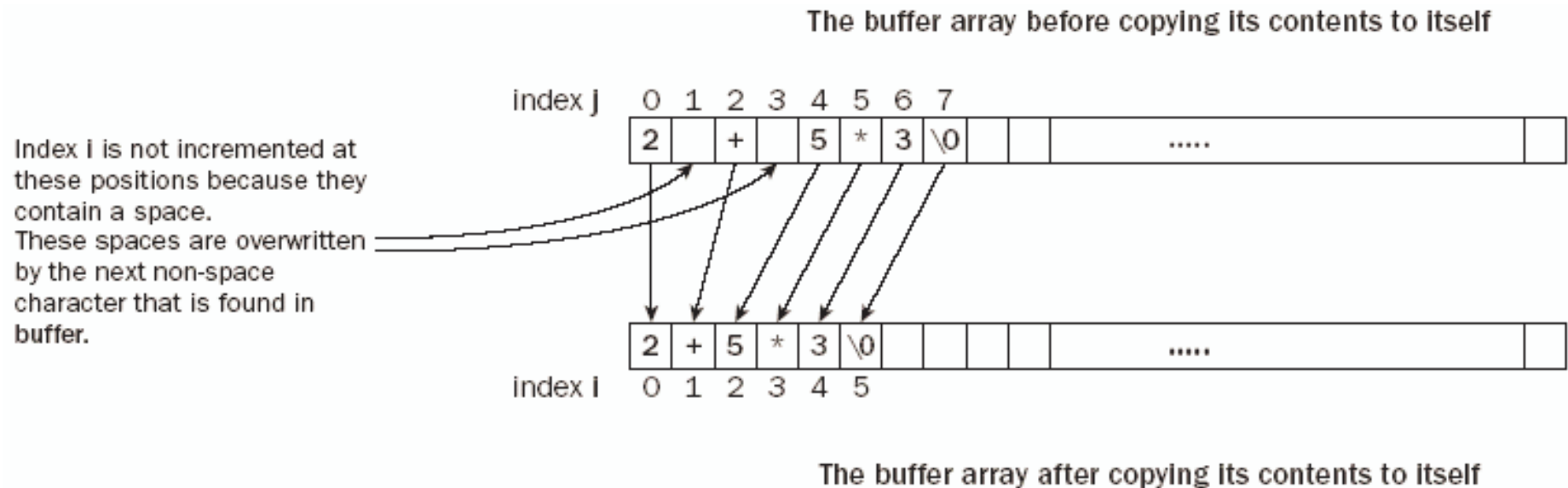


Figure 6-2

P.295

```
// Function to eliminate spaces from a string
void eatspaces(char* str)
{
    int i = 0;        // 'Copy to' index to string
    int j = 0;        // 'Copy from' index to string

    while ((* (str + i) = * (str + j++)) != '\0')
        if (* (str + i) != ' ')
            i++;
    return;
}
```

- Now, we obtain an expression with no embedding spaces.

Step 2: Evaluating an Expression

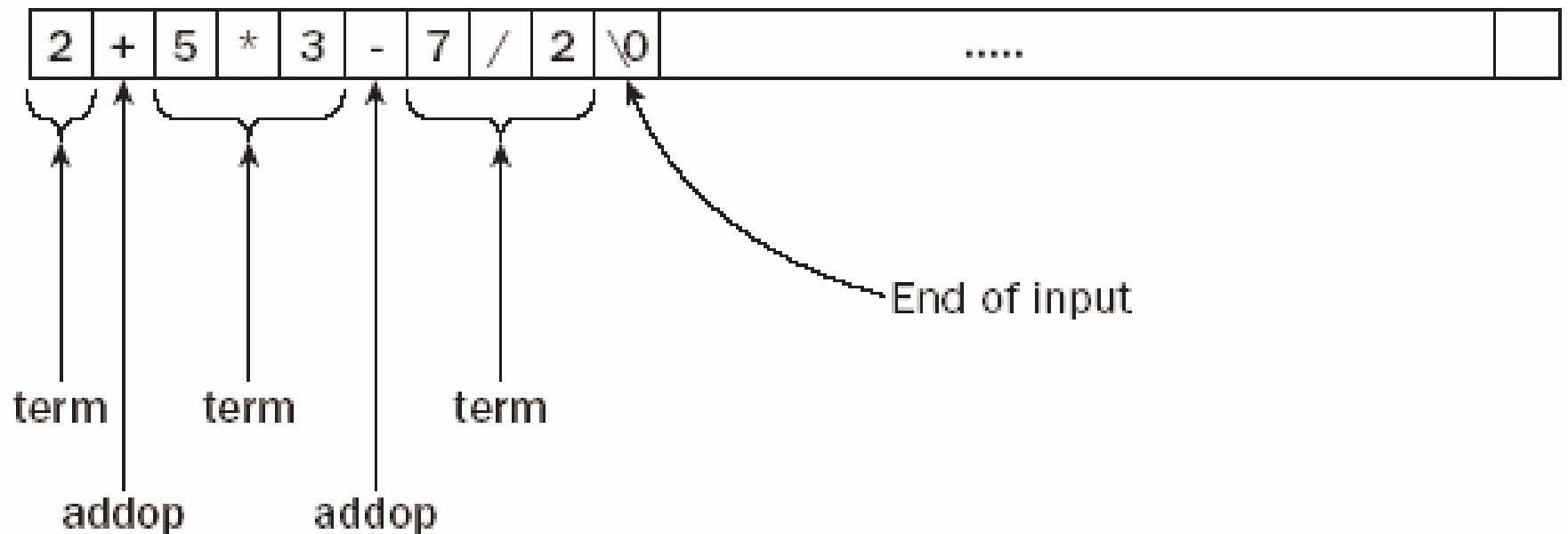


Figure 6-3

Breaking Down an Expression into Terms and Numbers

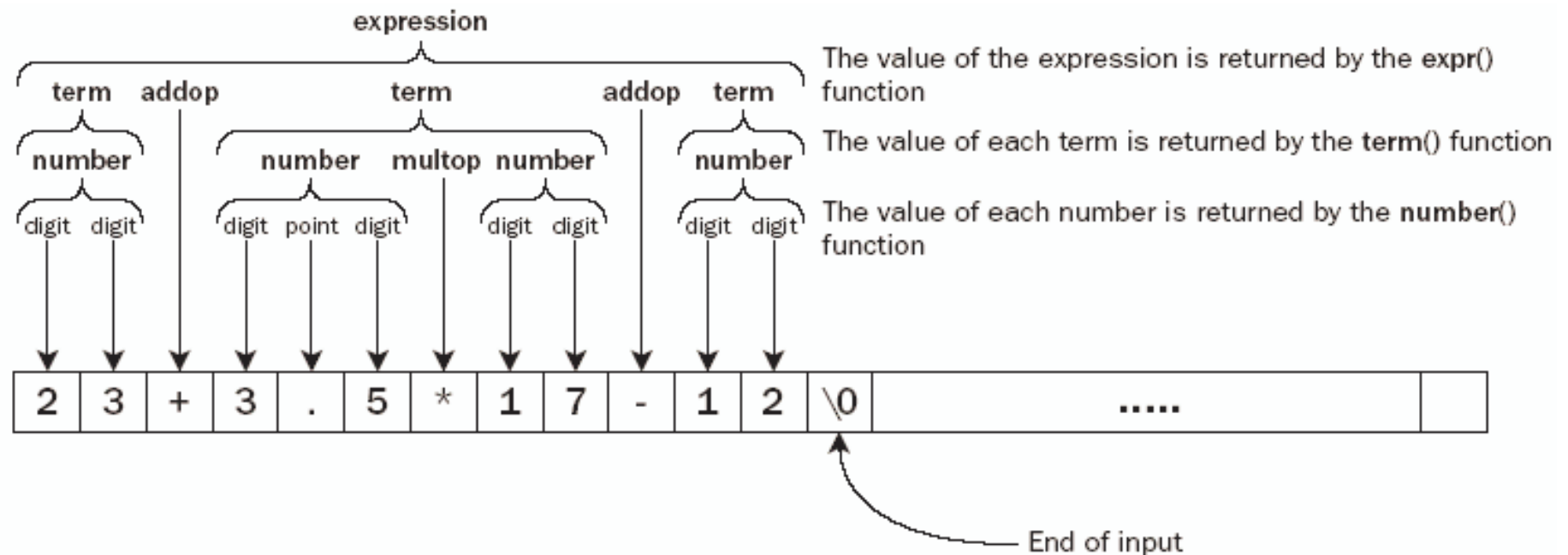
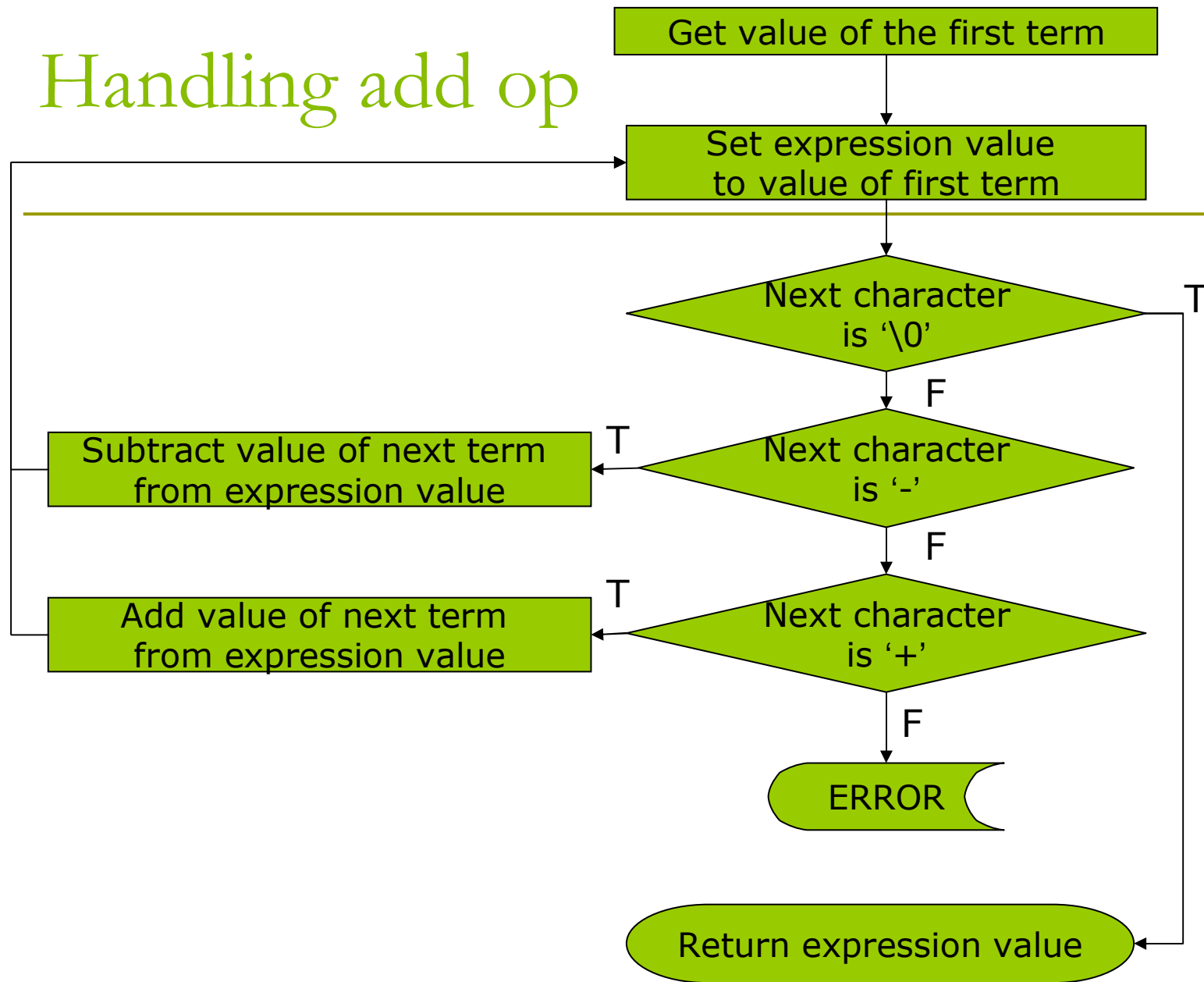


Figure 6-4

Handling add op



```
double expr(char* str)
{
    double value = 0.0;
    int index = 0;

    value = term(str, index);

    for (;;)
    {
        switch (*(str + index++))
        {
            case '\0':
                return value;
            case '-':
                value -= term(str, index);
            case '+':
                value += term(str, index);
            default:
                cout << endl << "Arrrrgh!*#!! There's an error" <<
endl;
                exit(1);
        }
    }
}
```

Getting the value of a Term

□ P.298

Analyzing a Number

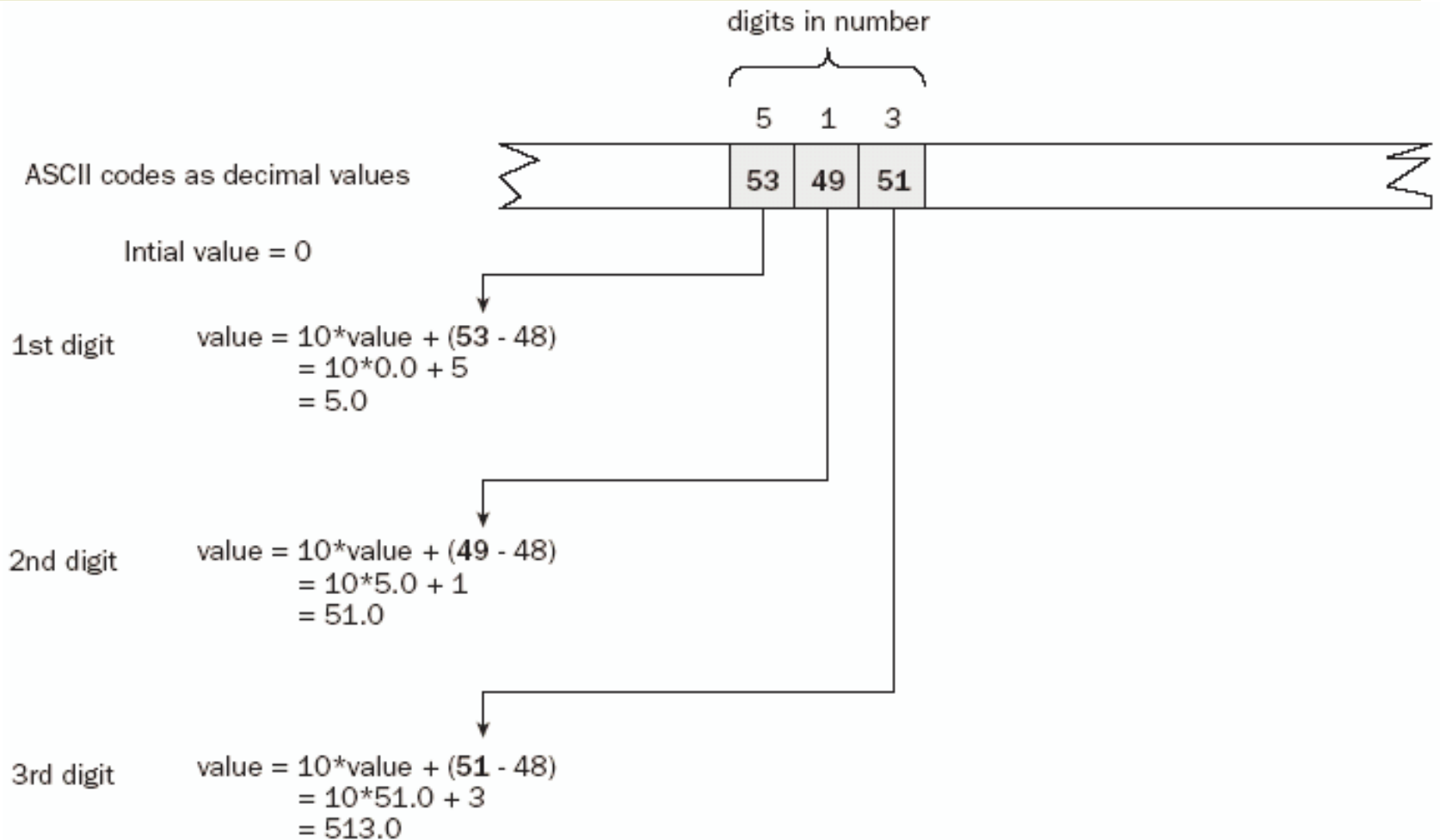


Figure 6-6


```
double number(char* str, int& index)
{
    double value = 0.0;



---


    while (isdigit(*(str + index)))
        value = 10 * value + ( *(str + index++) - '0');

    if (*(str + index) != '.')
        return value;

    double factor = 1.0;
    while (isdigit(*(str + (++index))))
    {
        factor *= 0.1;
        value = value + ( *(str + index) - '0') * factor;
    }

    return value;
}
```

Handling the fractional part after the decimal point

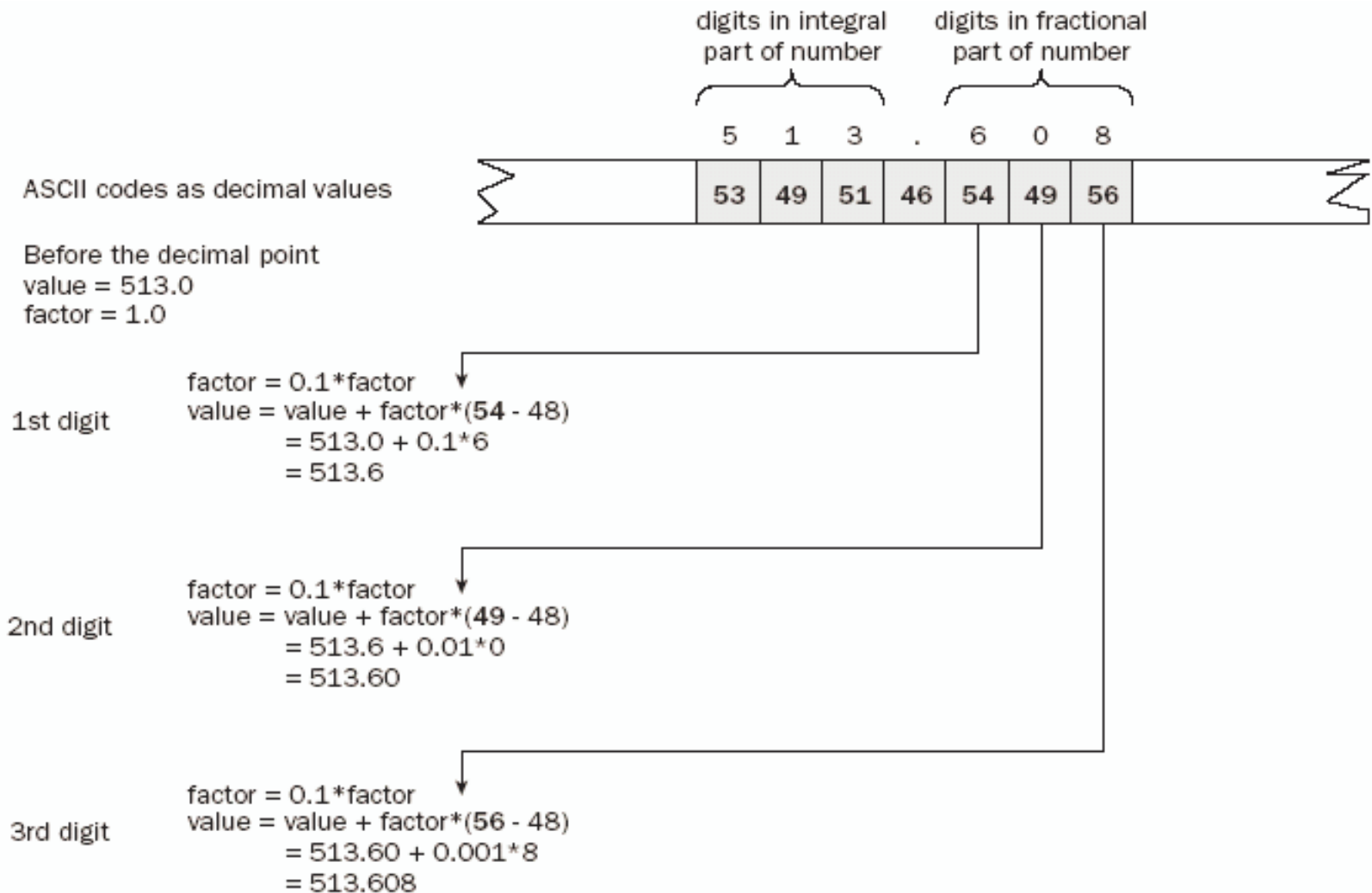


Figure 6-7

Putting the Program Together

□ P.302 Ex6_09.cpp

- `#include <iostream> // For stream input/output`
- `#include <cstdlib> // For exit() function`
- `#include <cctype> // For isdigit() function`

□ Use `cin.getline()` so that the input string can contain spaces.

- See P.167

Extending the Program

- Let us try to extend it so that it can handle parentheses:
 - $2 * (3 + 4) / 6 - (5 + 6) / (7 + 8)$

- Idea: treat an expression in parentheses as just another number.
 - P.304
 - `expr()` recursively calls itself
 - `expr()` - `term()` - `number()` - `expr()`
 - The string pointed by `p_substr` is allocated in `extract()`, and must be freed as an array/

extract ()

□ Extract a substring between parentheses

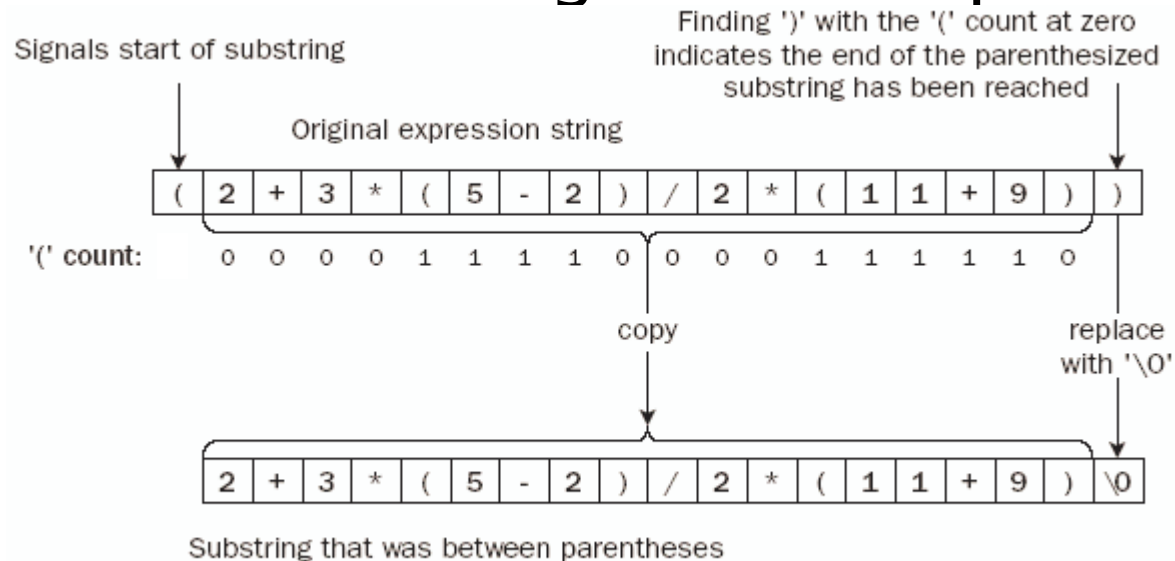


Figure 6-8

□ P.306

- Utilize `strcpy_s ()` which is defined in `<cstring>` header file