CS2313 Computer Programming

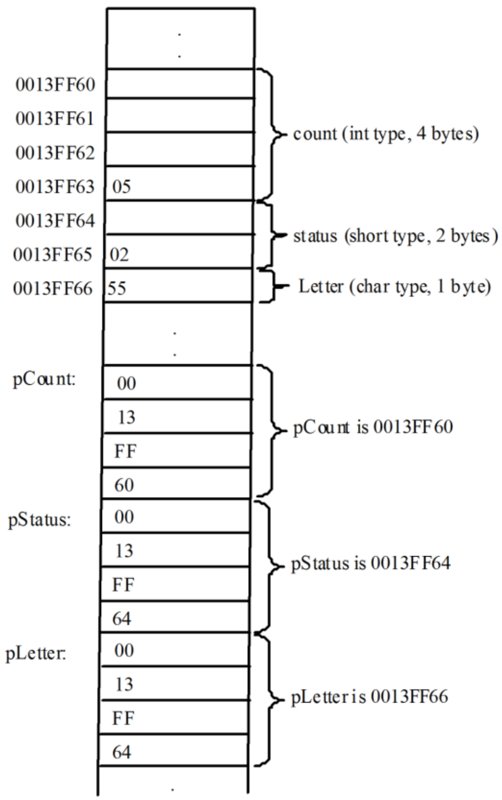
LT11 – Pointer/IO/Recursion

香港城市大學
City University of Hong Kong
專業 創新 胸懷全球
Professional · Creative
For The World

Outlines

- Revision on Pointer
  - Pointer Arithmetic
- I/O
  - Stream
  - Open File
  - File I/O
  - Error Handling
- Recursion

Pointer



int count= 5;
short status = 2;
char letter = 'A';

int * pCount = &count;
char * pLetter = &letter;

pCount = &count;

&: address operator
&count: the address of count

*: dereference operator
*pCount: value pointed to by pCount

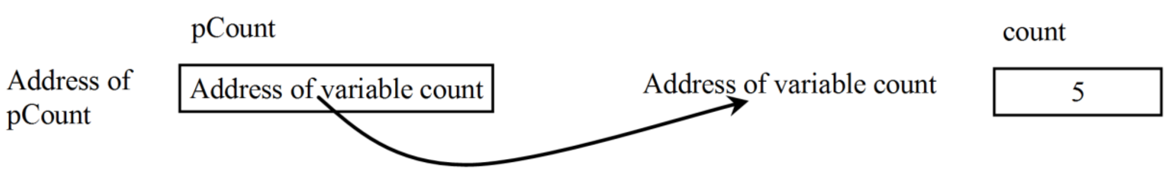
Declare a Pointer

Pointer, like normal variable has a type, its type is determined by the type of variable it points to.

dataType* pVarName;

Each variable being declared as a pointer must be preceded by an asterisk (*). The following statement declares a pointer variable named pCount that can point to an int varaible.

int* pCount; pCount=&count;



Dereferencing

Get access to the value of the variable pointed to by the pointer

*pointer

For example, you can increase count using

count++; // direct reference, increment the value in count by 1

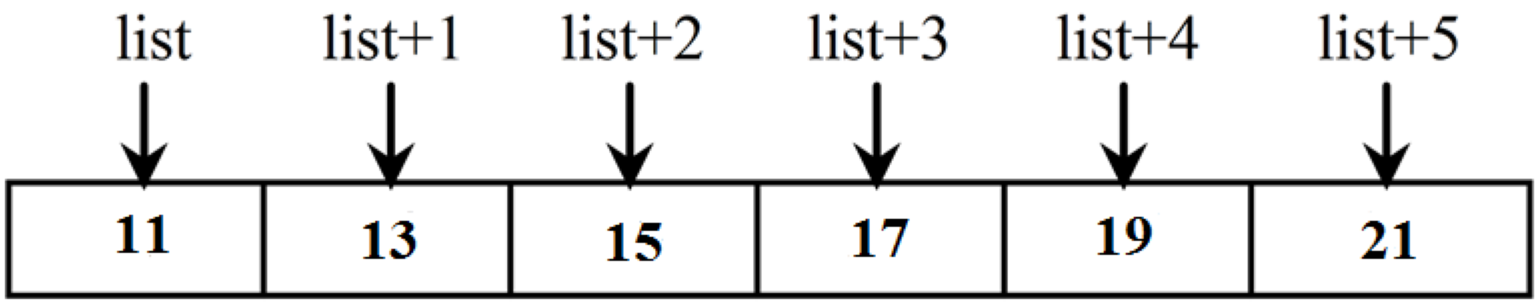
or

(*pCount)++; // indirect reference, the value in the memory pointed to by pCount is incremented by 1

Arrays and Pointers

An array variable without a bracket and a subscript represents the starting address of the array.
An array variable is essentially a pointer.

int list[6] = {11, 13, 15, 17, 19, 21};




```
cout<<*list;       //11 is printed;
cout<<(*list+1); //12 is printed;
cout<<*(list+1); //13 is printed;
```

Function-
array parameter or pointer parameter

```
void m(int list[], int size)
```

can be replaced by

```
void m(int *list, int size)
```

```
void m(char c_string[])
```

can be replaced by

```
void m(char *c_string)
```

Dynamic Memory Allocation

```
int* result = new int[n]; // Allocate

delete [] result; // Deallocate

int* p = new int; // Allocate

delete p; // Deallocate
```

Array and Pointer

```
int main()
{
    int values[5] = { 0,0,0,0,0 };
    for (int i = 1; i < 5; i++)
    {
        values[i] = i + values[i - 1];
    }
    values[0] = values[1] + values[4];

    return 0;
}
```

What are the values in values array?

11, 1, 3, 6, 10

Program with pointer arithmetic

```c
int main()
{
    int values[5] = { 0,0,0,0,0 };
    int *p = values;
    for (int i = 1; i < 5; i++)
    {
        *(p + 1) = i + (*p);
        p++;
    }
    values[0] = values[1] + values[4];

    return 0;
}
```

p points to the first element

dereference

p points to the next element

Example-reverse an array

- Task: write a program to reverse an array
  - Input: an array with 6 elements { 1, 2, 3, 4, 5, 6 }
  - Output: an array with 6 elements {6, 5, 4, 3, 2, 1}
  - The pointer **arithmetic** is required to be used
- Solution
  - Pointer: int *p…
  - Dereference: *p
  - Pointer++, --: p++, p--

Example-reverse an array

```
#define N 6

int main()
{
    int list1[] = { 1, 2, 3, 4, 5, 6 };
    int *list2;
    list2 = new int[N];

    reverse(list1, list2);

    for (int i = 0;i < N;i++)
    {
        cout << list2[i]<<endl;
    }

    delete []list2;
}
```

```
void reverse(int* list1, int *list2)
{
    int * p1 = list1;
    int * p2 = list2 + N-1;

    for (int i = 0; i < N; i++)
    {
        *p2 = *p1;
        p2--;
        p1++;
    }
}
```

Example-dynamic memory allocation

```
int main()
{
    int *list;
    int n;

    cin >> n;

    list = new int[n];

    for (int i = 0; i < n; i++)
    {
        cin >> list[i];
    }

    delete []list;
}
```

We can use the variable to initialize the array size!

If you still need to use the array *list* (return list in the function), do not delete the array.

Pointer (Function – count)

```
int count(char *s, char c)
{
   int occurrence=0;

   for (char * pi=s; *pi!='\0'; pi++){
      if (*pi==c)
         occurrence++;
   }
   return occurrence;
}
```

Pointer (Function – count)

```
void main(){
   char str[]="Hong Kong is a very good place to live";
   int count1 = count(str, 'o');
   cout << "count = " <<count1<< " \n";
}
```

Pointer Arithmetic

```
int main()
{
    int value = 7;
    int *ptr = &value;

    std::cout << ptr << '\n';
    std::cout << ptr + 1 << '\n';
    std::cout << ptr + 2 << '\n';
    std::cout << ptr + 3 << '\n';

    return 0;
}
```

```
012FF764
012FF768
012FF76C
012FF770
Press any key to continue . . .
```

each of these addresses differs by 4. This is because an integer is 4 bytes on the machine.

Pointer Arithmetic

```
int main()
{
    short value = 7;
    short *ptr = &value;

    std::cout << ptr << '\n';
    std::cout << ptr + 1 << '\n';
    std::cout << ptr + 2 << '\n';
    std::cout << ptr + 3 << '\n';

    return 0;
}
```

```
010EF9E4
010EF9E6
010EF9E8
010EF9EA
Press any key to continue . . .
```

each of these addresses differs by 2. This is because a short is 2 bytes on the machine.

I/O-Outlines

- Stream
- Open File
- File I/O
- Error Handling

I/O-Outcomes

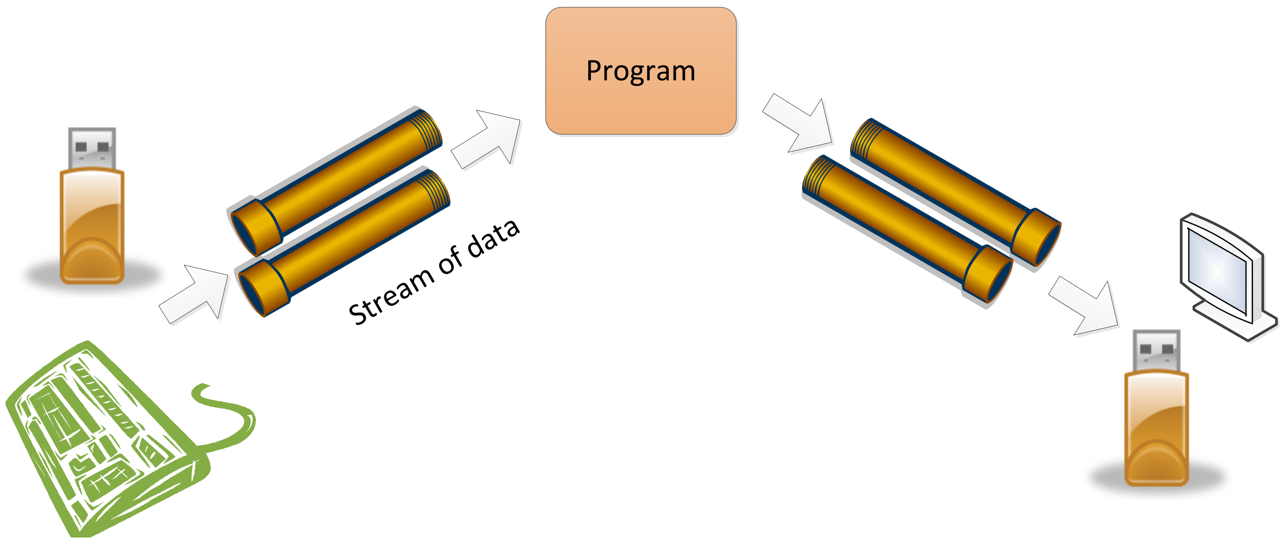
- Able to read a file
- Write data to file
- Handle file

File I/O vs. Console I/O

- “Console” refers to “keyboard + screen”.
- Keyboard input and screen output are volatile.
- Input file can be used again and again.
- Useful for debugging especially when volume of data is huge.
- Allow off-line processing.
- Output file retains the results after execution.

Basic I/O – Keyboard and Screen

- Program read input from keyboard (console) or disk storage (file) and write data to screen (console) or disk storage(file).
- Sequence of inputs is conceptually treated as an object called “Stream”.
- Stream – a flow (sequence) of data.
- Input stream – a flow of data into your program.
- Output stream – a flow of data out of your program.



Stream

- Predefined console streams in C++
  - #include <iostream>
  - `cin : input stream physically linked to the keyboard`
  - `cout: output stream physically linked to the screen`

- File stream class in C++
  - #include <fstream>
  - `ifstream:  stream class for file input`
  - `ofstream:  stream class for file output`

- To declare an objects of class ifstream or ofstream, use
  - `ifstream fin;`
  - `ofstream fout;`

ifstream

- To declare an ifsteam object
  - `ifstream fin;`
- To open a file for reading
  - `fin.open("infile.dat");`
- To read the file content
  - `fin >> x;   //x is a variable`
- To close the file
  - `fin.close();`

`ofstream`

- To declare an ofsteam object
  - `ofstream fout;`
- To open a file for writing
  - `fout.open("myfile.dat") ;`
- To write something to the file
  - `fout << x;   //x is a variable`
- To close the file
  - `fout.close();`
- PS: `fin.open()` and `fout.open()` refer to different functions

Examples

```
#include <fstream>
using namespace std;
void main(){
   ifstream fin;
   ofstream fout;
   int x,y,z;
   fin.open("input.txt");
   fout.open("output.txt");
   fin >>x>>y>>z;
   fout << "The sum is "<<x+y+z;
   fin.close();
   fout.close();
}
```

3 4 7

The sum is 14

Detecting end-of-file

- Member function `eof` returns true if and only if we try to read from the input file which has no more data
  - Only for objects of class `ifstream`

  E.g. `fin >> x;`

  `if (fin.eof())` …
- The expression `fin >> x` has value 0 if `fin` has no more data

  E.g.
```
while (fin>> x)
  {...}
```

Examples: File Dump (Integer Only)

```
#include <iostream>
#include <fstream>
using namespace std;
void main(){
   ifstream fin;
   int x;
   fin.open("input.txt");
   while (!fin.eof()){
      fin >> x;
      cout <<x<<" ";
   }
}
```

Examples: File Dump (Integer Only)

```
#include <iostream>
#include <fstream>
using namespace std;
void main(){
   ifstream fin;
   int x;
   fin.open("input.txt");
   while (fin >> x){
      cout <<x<<" ";
   }
}
```

return 0 if fin has no more data

Detecting I/O Failures

- Member function `fail()` returns `true` if and only if the previous I/O operation on that stream fails.
- E.g. file not exists when opening an input stream.
- PS: one may call function `exit()` when an I/O operation fails to abort the program execution.
- the argument in `exit()` is returned to the calling party -- usually the OS.

Examples

```
#include <iostream>
#include <fstream>
Using namespace std;

void main(){
  ifstream in1, in2;

  in1.open("infile1.dat");
  in2.open("infile2.dat");
  if (in1.fail()) {
     cout << "Input file 1 opening failed.\n";
     exit(1);     // 1 stands for error
     }
   ...
```

Reference Only: I/O Re-directions

- A facility offered by many OS's.
- Allows the program input and output to be redirected from/to specified files.
- E.g. suppose you have an executable file hello.exe. If you type:

```
hello > outfile1.dat
```

- in the MSDOS prompt, the output is written to the file outfile1.dat instead of the screen.
- Similarly, `hello < infile1.dat` specifies that the input is from infile1.dat instead of the keyboard.

Summary

- Beside reading and writing data from and to console, program can read and write data from and to file.

- ifstream and ofstream are two classes defined in <fstream>.

- File must be open before access and close after access.
  - `fin.open("filename");`
  - `fin.close();`

- File I/O is similar to console I/O.
  - `cin >> x;`
  - `fin >> x;`

Recursion

- A recursive function is a function that calls itself.
- In some problems, it may be natural to define the problem in terms of the problem itself.
- Recursive functions can be useful in solving problems that can be broken down into ***smaller or simpler sub-problems*** of the **same type**.
- A base case should eventually be reached, at which time the breaking down (recursion) will stop.

Example 1:
Problem of Recursive Nature (1)

The factorial function

```
6! = 6 * 5 * 4 * 3 * 2 * 1
```

We could write:

```
6! = 6 * 5!
```

Example 1:
Problem of Recursive Nature (2)

In general, we can express the factorial function on the last slide as follows:

```
n! = n * (n-1)!
```

Is this correct? Well… almost.

The factorial function is only defined for *positive* integers. So we should be a bit more precise:

```
n! = 1    (if n is equal to 1)
n! = n * (n-1)!    (if n is larger than 1)
```

Example 1:
Problem of Recursive Nature (3)

The C++ equivalent of this definition:

```
int fac(int numb){
   if(numb<=1)
        return 1;
   else
      return numb * fac(numb-1);
}
```

Example 1:
Problem of Recursive Nature (4)

- Assume the number typed is 3, that is, numb=3.

```
fac(3) :

 3 <= 1 ?            No.
 fac(3) = 3 * fac(2)

   fac(2) :
      2 <= 1 ?           No.
      fac(2) = 2 * fac(1)

          fac(1) :
             1 <= 1 ?    Yes.
             return 1

       fac(2) = 2 * 1 = 2
       return fac(2)

 fac(3) = 3 * 2 = 6
 return fac(3)

 fac(3)  has the value 6
```

```
int fac(int numb){
   if(numb<=1)
       return 1;
   else
       return numb * fac(numb-1);
}
```

Example 1: Problem of Recursive Nature (5)

- For certain problems (such as the factorial function), a recursive solution often leads to short and elegant code. Compare the recursive solution with the iterative solution:

Recursive solution

```
int fac(int numb){
  if(numb<=1)
     return 1;
  else
    return numb*fac(numb-1);
}
```

Iterative solution

```
int fac(int numb){
  int product=1;
  while(numb>1){
    product *= numb;
    numb--;
   }
   return product;
}
```

Recursion

If we use iteration, we must be careful, not to create an infinite loop by accident:

```
for(int incr=1; incr!=10;incr+=2)
   ...
```

Oops!

```
int result = 1;
while(result >0){
   ...
   result++;
}
```

Oops!

Recursion

Similarly, if we use recursion, we must be careful not to create an infinite chain of function calls:

```
int fac(int numb){
    return numb * fac(numb-1);
}
```

Oops!

No termination condition

Or:

```
int fac(int numb){
    if (numb<=1)
        return 1;
    else
        return numb * fac(numb+1);
}
```

Oops!

Recursion

We must always make sure that the recursion *bottoms out*:

- A recursive function must contain at least one non-recursive branch.
- The recursive calls must eventually lead to a non-recursive branch.

Recursion

- Recursion is one way to decompose a task into smaller subtasks. At least one of the subtasks is a smaller example of the same task.
- The smallest example of the same task has a non-recursive solution.

Example: The factorial function

```
n! = n * (n-1)! and 1! = 1
```

Direct Computation Method

- Fibonacci numbers:

  `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...`

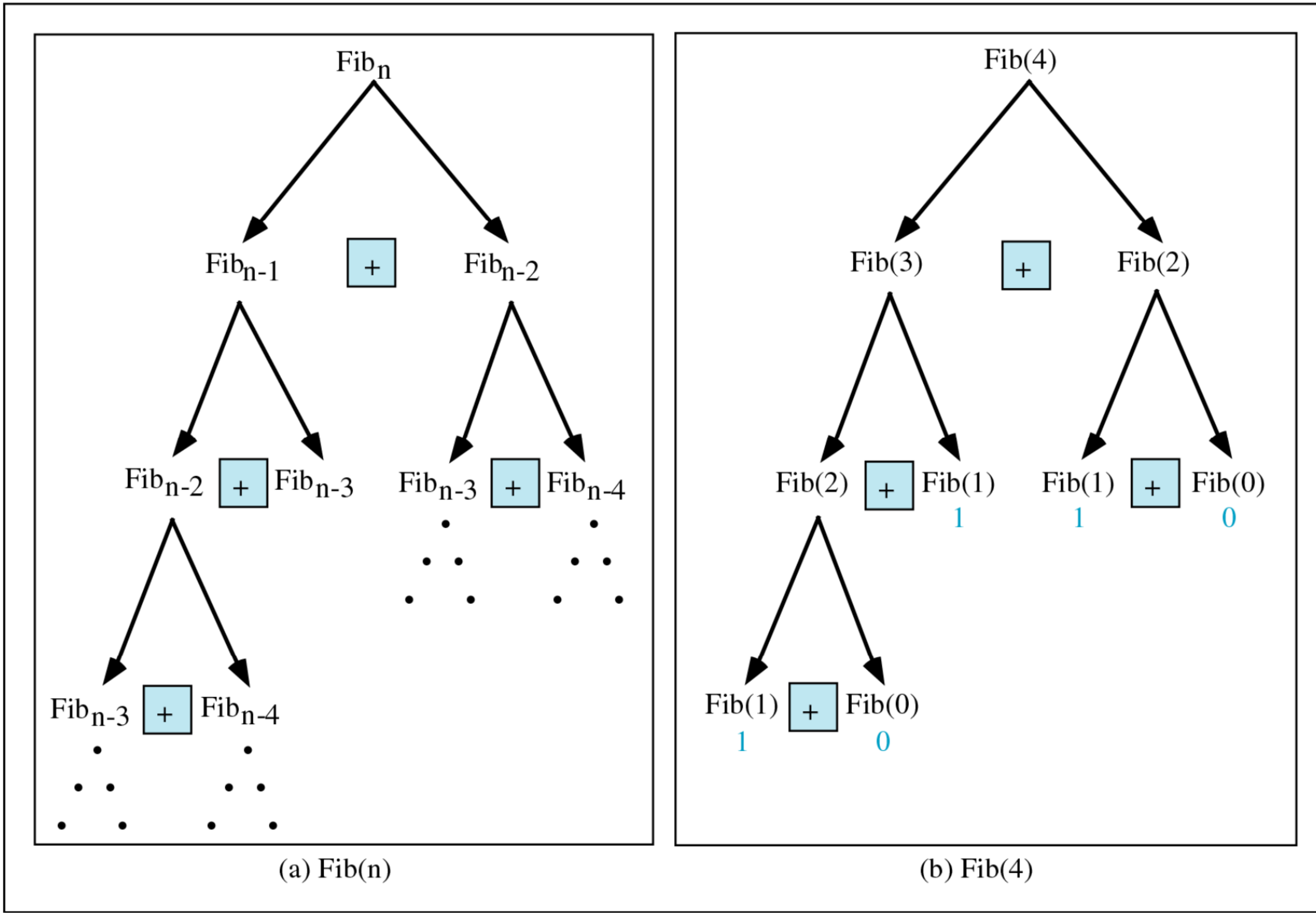
  where each number is the sum of the preceding two.

- Recursive definition:
  - **`F(0) = 0;`**
  - **`F(1) = 1;`**
  - **`F(number) = F(number-1)+ F(number-2);`**

Example 2: Fibonacci numbers

```
//Calculate Fibonacci numbers using recursive function.
//A very inefficient way, but illustrates recursion well
int fib(int number)
{
    if (number == 0) return 0;
    if (number == 1) return 1;
    return (fib(number-1) + fib(number-2));
}


int main(){// driver function
     int inp_number;
     cout << "Please enter an integer: ";
     cin >> inp_number;
     cout << "The Fibonacci number for "<< inp_number
      << " is "<< fib(inp_number)<<endl;
       return 0;
}
```



Fib$_n$
Fib$_{n-1}$
+
Fib$_{n-2}$
Fib$_{n-2}$
+
Fib$_{n-3}$
Fib$_{n-3}$
+
Fib$_{n-4}$
Fib$_{n-3}$
+
Fib$_{n-4}$
Fib(4)
Fib(3)
+
Fib(2)
Fib(2)
+
Fib(1)
1
Fib(1)
1
+
Fib(0)
0
Fib(1)
1
+
Fib(0)
0

(a) Fib(n)

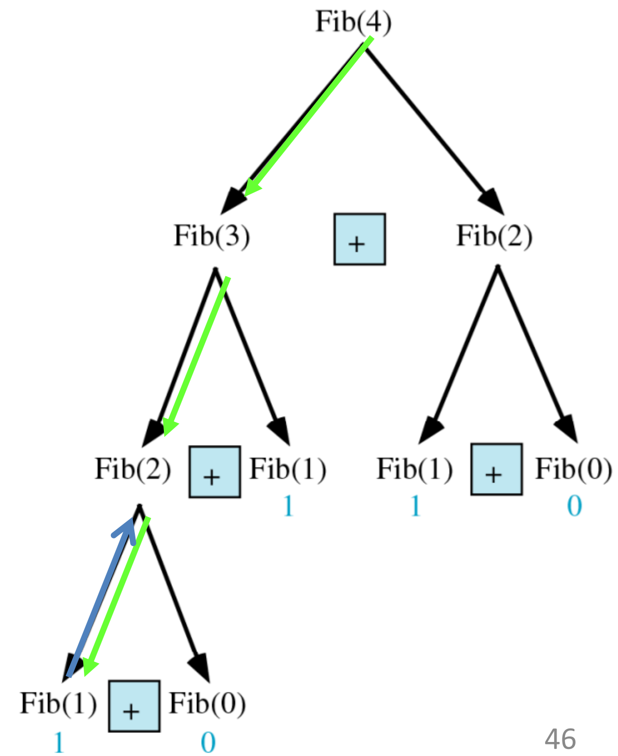
(b) Fib(4)

Trace a Fibonacci Number

- Assume the input number is 4, that is, num=4:

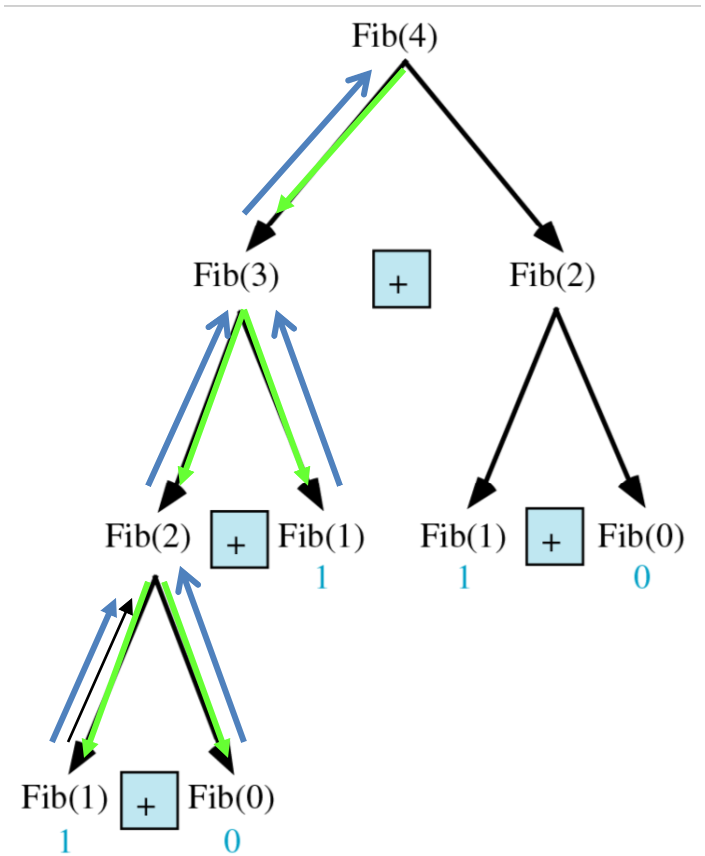
```
fib(4):
    4 == 0 ? No;    4 == 1?  No.
    fib(4) = fib(3) + fib(2)
    fib(3):
        3 == 0 ? No; 3 == 1? No.
        fib(3) = fib(2) + fib(1)
        fib(2):
            2 == 0? No; 2==1? No.
            fib(2) = fib(1)+fib(0)
           fib(1):
               1== 0 ? No; 1 == 1? Yes.
            fib(1) = 1;
            return fib(1);
```

```
int fib(int num)
{
    if (num == 0) return 0;
    if (num == 1) return 1;
    return
      (fib(num-1)+fib(num-2));
}
```



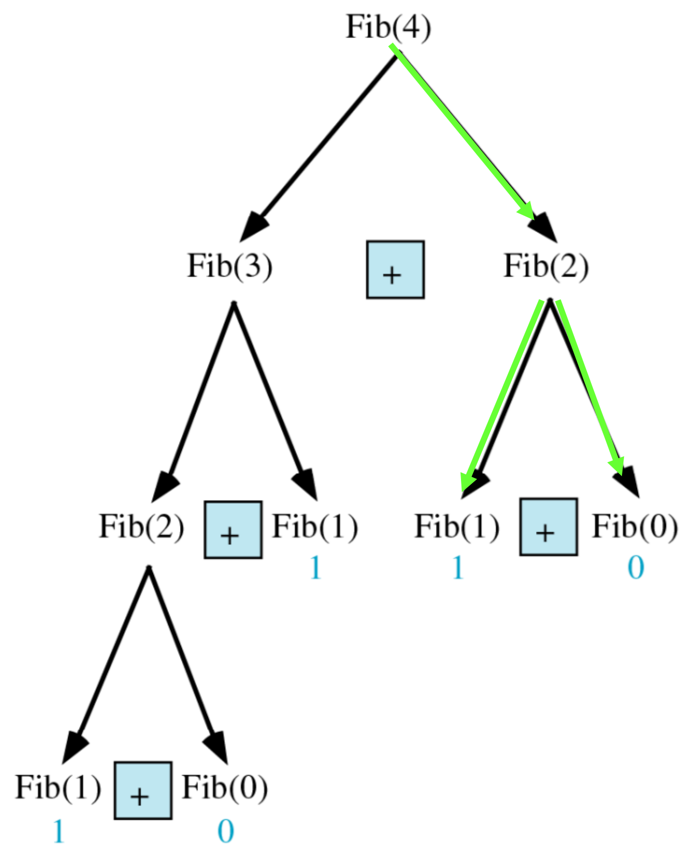
Trace a Fibonacci Number

```
    fib(0):
        0 == 0 ?  Yes.
     fib(0) = 0;
        return fib(0);
  fib(2) = 1 + 0 = 1;
  return fib(2);
fib(3) = 1 + fib(1)
 fib(1):
     1 == 0 ? No; 1 == 1? Yes
     fib(1) = 1;
     return fib(1);
fib(3) = 1 + 1 = 2;
return fib(3)
```



Trace a Fibonacci Number

```
fib(2):
    2 == 0 ? No; 2 == 1?No.
    fib(2) = fib(1) + fib(0)
    fib(1):
       1== 0 ? No; 1 == 1?  Yes.
    fib(1) = 1;
    return fib(1);
     fib(0):
        0 == 0 ?   Yes.
    fib(0) = 0;
       return fib(0);
     fib(2) = 1 + 0 = 1;
     return fib(2);
 fib(4) = fib(3) + fib(2)
          = 2 + 1 = 3;
 return fib(4);
```



Example 3: Fibonacci number w/o recursion

```
//Calculate Fibonacci numbers iteratively
//much more efficient than recursive solution

int fib(int n)
{
   int f[100];
   f[0] = 0; f[1] = 1;
     for (int i=2; i<= n; i++)
          f[i] = f[i-1] + f[i-2];
   return f[n];
}
```