What is a "Better" Program?

C++ Object Oriented Programming

Pei-yih Ting

NTOU CS

軟體的特性

✧ 軟體之所謂軟…因為沒有 "硬性" 不可變、不可挑戰的規則
  * 好處: 彈性很大, 山不轉路轉, 沒有標準答案, 正常運作就好…
  * 壞處: 很多小問題合在一起不斷放大, 到處藏污納垢, 沒有標準答案, 不知道到底對了沒有

✧ 解決方法
  * **Coding styles**
  * test-driven
  * 元件化
  * 模型化 (資料結構, 演算法, 物件化, 軟體模式)

Source Code is the Primary Document

✧ Jack Reeves, C++ Journal, 1992, "What is Software Design?"

  "After reviewing the software development life cycle as I understood it, I concluded that the only software **documentation** that actually seems to satisfy the criteria of an engineering design is the **source code listings**."

✧ The **design** of a software project is an abstract concept:
  * It has to do with the overall shape and structure of the program as well as the detailed shape and structure of each module, class, and method.
  * It can be represented by many different diagrams and media, but its final embodiment is the source code.

✧ **Source code is the design**

Goals

✧ 透過一些基本的編碼規則，我們可以寫出一個 "好" 一點的 C 程式

✧ 除了正確性之外，程式短一點?? 執行快一點???

✧ "好" ?   (in terms of test, debug, review, and extension)
  1. 容易了解，沒有邏輯上不緊密結合的資料變數或是敘述
  2. Self-explaining
  3. 和觀念上的運作模型一致
  4. 容易修改，不容易改錯
  5. 沒有容易錯誤的語法

✧ 正確性無關：以下給你一個很簡單的例子，共有七個版本，執行結果都是正確的

Version 1

```
01 #include <stdio.h>
02
03 void main()
04 {
05    int d[] = {12, 3, 37, 8, 24, 15, 5, 33};
06    int n = 8;
07    int *d1, *d2;
08    int *p;
09    int *e;
10
11    d1 = d;
12    d2 = d+n;
13    while (d1<d2)
14    {
15       p = d1;
16       e = d1 + 1;
17       while (e<d2)
18       {
19          if (*e<*p) p = e;
20          e++;
21       }
22       n = *p;
23       *p = *d1;
24       *d1 = n;
25       d1++;
26    }
27    printf("Sorted data:\n");
28    d1 = d;
29    while (d1<d2)
30       printf(" %d", *d1++);
31    printf("\n");
32 }
```

Execution Results

Sorted data:
   3 5 8 12 15 24 33 37

由小至大按順序排列

What is this program doing?

Initial view

✧ Input array initialized with unordered integers

✧ Two layers of while loops

✧ Some pointers to the elements of the array

✧ Another while loop for output the results

Don't like it!!??

✧ Pointers

✧ Generic while loops

✧ Variable names (identifier means nothing)

✧ Deep control structures

✧ Looks like a snippet of low level assembly instructions

Remove Unnecessary Pointers

✧ Pointers are sophisticated and sometimes inevitable, but not always.

✧ In the case of accessing memory blocks, pointers are extraneous, use array whenever possible.

✧ Array has much better semantic meaning than the generic pointer dereferencing.

```
int array[100];              int array[100];
int *ptr=array;              int i;
int i, sum = 0;             int sum = 0;
…                           …
for (i=0; i<100; i++)        for (i=0; i<100; i++)
   sum += *ptr++;              sum += array[i];
```

Version 2

```
01 #include <stdio.h>                              17        j = j + 1;
02                                                 18      }
03 void main()                                     19      j = d[k];
04 {                                               20      d[k] = d[i];
05    int d[] = {12, 3, 37, 8, 24, 15, 5, 33};     21      d[i] = j;
06    int n = 8;                                   22      i = i + 1;
07    int i, j, k;                                 23    }
08                                                 24    printf("Sorted data:\n");
09    i = 0;                                       25    i = 0;
10    while (i<n)                                  26    while (i<n)
11    {                                            27    {
12      k = i;                                     28      printf(" %d", d[i]);
13      j = i + 1;                                 29      i = i + 1;
14      while (j<n)                                30    }
15      {                                          31    printf("\n");
16        if (d[j]<d[k]) k = j;                    32 }
```

9

Flowchart of the Program
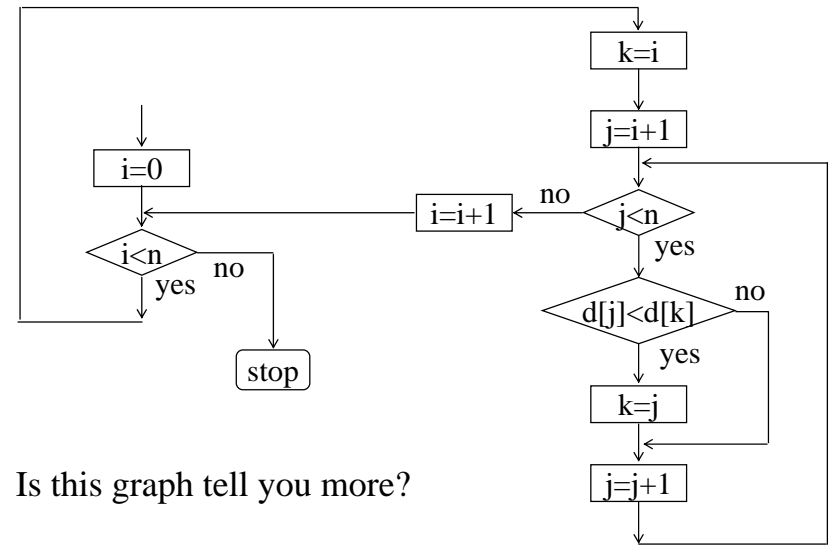
Is this graph tell you more?

10

Meaningful Identifiers

✧ A program is composed with a language. Just like any language in your daily life, language itself should **tell good stories** when used properly.

✧ Why does the version 1 or version 2 program look like gibberish to a well trained programmer?

✧ Are the **identifiers** used meaningful??
   e.g.

           Hw ds Jhn lk th stk?
   or
        How does John like the steak?

11

Version 3

```
01 #include <stdio.h>                                    17    {
02                                                       18        if (data[j]<data[min]) min = j;
03 void main()                                           19        j = j + 1;
04 {                                                      20    }
05    int data[] = {12, 3, 37, 8, 24, 15, 5, 33};        21    swapTmp = data[min];
06    int ndata = sizeof(data) / sizeof(int);            22    data[min] = data[i];
07    int i, j;        avoid magic constants             23    data[i] = swapTmp;
08    int min;                                           24    i = i + 1;
09    int swapTmp;                                        25  }
10                                                       26
11    i = 0;                                             27  printf("Sorted data:\n");
12    while (i<ndata)                                    28  i = 0;
13    {                                                  29  while (i<ndata)
14      min = i;                                         30  {
15      j = i + 1;                                        31    printf(" %d", data[i]);
16      while (j<ndata)                                  32    i = i + 1;
                                                         33  }
                                                         34  printf("\n");
                                                         35 }
```

12

Advanced View of the Codes

Initial view

✧ Input array initialized with unordered integers

✧ Two layers of while loops

✧ Some pointers to the elements of the array

✧ Another while loop for output the results

Is it changing?

✧ Input array initialized with unordered integers

✧ Two layers of while loops, the outer one prepares ndata sub-arrays, the inner one goes through each sub-array to find something minimum

✧ A snippet of memory swapping code

✧ Another while loop for output the results

More Meaningful Language Construct

✧ While loop is the most generic repetition construct in C language
  initialize the loop condition (let's not even think of *goto*)
  while (condition)
  {
       …
  }

  the condition might change inside the loop

✧ When you see this construct in a program, you expect some sort of job repetition, maybe an easy one or a complex one.

✧ For loop is a more semantically specific repetition construct in C language --- repeat for a predetermined number of times
  for (i=0; i<count; i++)
  {
       …
  }

Version 4

```
01 #include <stdio.h>
02
03 void main()
04 {
05    int data[] = {12, 3, 37, 8, 24, 15, 5, 33};
06    int ndata = sizeof(data) / sizeof(int);
07    int i, j;
08    int min;
09    int swapTmp;
10
11    for (i=0; i<ndata; i++)
12    {
13      min = i;
14      for (j=i+1; j<ndata; j++)
15      {
16        if (data[j]<data[min]) min = j;
17      }
18      swapTmp = data[min];
19      data[min] = data[i];
20      data[i] = swapTmp;
21    }
22
23    printf("Sorted data:\n");
24    for (i=0; i<ndata; i++)
25      printf(" %d", data[i]);
26    printf("\n");
27 }
```

Code That Further Illustrates Itself

✧ **Function** is a powerful construct to **abstract** ideas, not just a utility for saving your typing time or some sacred code-reuse purpose.

> --- Version 5

✧ Construct of **"loop inside a loop"** is somehow beyond the concrete control of human mind. A single layer of "loop" is better for most people to visualize in mind.

> --- Version 6

Version 5

```
01 #include <stdio.h>
02
03 void swap(int *, int *);
04 void printArrayContents(int [], int);
05
06 void main()
07 {
08    int data[] = {12, 3, 37, 8, 24, 15, 5, 33};
09    int ndata = sizeof(data) / sizeof(int);
10    int i, j;
11    int min;
12
13    for (i=0; i<ndata; i++)
14    {
15       min = i;
16       for (j=i+1; j<ndata; j++)
17       {
18          if (data[j]<data[min]) min = j;
19       }
20       swap(&data[i], &data[min]);
21    }
22
23    printArrayContents(data, ndata);
24 }
25
26 void swap(int *x, int *y)
27 {
28    int tmp;
29    tmp = *x;
30    *x = *y;
31    *y = tmp;
32 }
```

Version 5 (cont'd)

```
33
34 void printArrayContents(int data[], int ndata)
35 {
36    int i;
37    printf("Sorted data:\n");
38    for (i=0; i<ndata; i++)
39       printf(" %d", data[i]);
40    printf("\n");
41 }
```

Version 6

```
01 #include <stdio.h>
02
03 void selectionSort(int[], int);
04 void findMinimumOfAnArray(int[], int);
05 void swap(int*, int*);
06 void printArrayContents(int[], int);
07
08 void main()
09 {
10    int data[] = {12, 3, 37, 8, 24, 15, 5, 33};
11    int ndata = sizeof(data) / sizeof(int);
12
13    selectionSort(data, ndata);
14    printArrayContents(data, ndata);
15 }
16
```
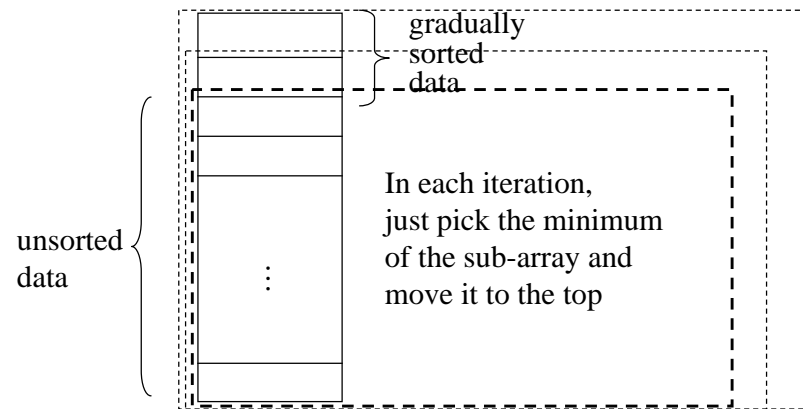
Version 6 (cont'd)

```
17 void selectionSort(int data[], int ndata)
18 {                    suitable level of details
19    int i;
20    for (i=0; i<ndata; i++)
21       putMinimalElementInPlace(&data[i], ndata-i);
22 }
23
24 void putMinimalElementInPlace(int data[], int ndata)
25 {
26    int i, min;
27
28    min = 0;
29    for (i=1; i<ndata; i++)
30    {
31       if (data[i]<data[min]) min = i;
32    }
33    swap(&data[0], &data[min]);
34 }
35
```

```
36 void swap(int *x, int *y)
37 {
38    int tmp;
39    tmp = *x;
40    *x = *y;
41    *y = tmp;
42 }
43
44 void printArrayContents(int data[], int ndata)
45 {
46    int i;
47    printf("Sorted data:\n");
48    for (i=0; i<ndata; i++)
49       printf(" %d", data[i]);
50    printf("\n");
51 }
```

Codes with a Conceptual Model

✧ Flowchart is no longer needed but definitely requires a conceptual model for the codes to work with.

gradually
sorted
data

In each iteration,
just pick the minimum
of the sub-array and
move it to the top

unsorted
data

⋮

Who is responsible of this task?

✧ The programmer or the program reader?

✧ When we read the version 1 of this program, there were little clues in the codes that told us directly what the program is doing.

✧ Although we figure out that this is a piece of code that implements the selection sort algorithm at last, it should not take the original programmer too much effort to produce a code snippet like version 6 and its corresponding conceptual model which tell directly the story of what the program is doing.

✧ A piece of code is to implement some engineering design, **simplicity** is the best engineering principle. Try your best to think and express ideas in an intuitive way.

Recursive Version

✧ Recursive version is often the most expressive form of the underlying algorithm.

```
void selectionSort(int data[], int ndata)
{
    putMinimalElementInPlace(data, ndata);
    if (ndata>2)
        selectionSort(&data[1], ndata-1);

}
```

Efficiency Issues

✧ Using expressive name for all identifiers makes the program much lengthier, easier to have typos, slow in composing the program.
  * Harddisk is cheap. Not necessary to think of space.
  * It is easier for compiler to detect typo than using x, y, z.
  * Typing should not be the bottleneck.
  * Expressive programs are easier to compose, maintain, and extend.

✧ Excessive function calls take CPU time to transfer arguments and to branch the control.
  * Let the compiler worry about it --- use inline function.

✧ Using dedicated variables for independent tasks looks like abusing memories.
  * Let the compiler worry about it.
  * Lesser bugs is a far bigger concern.

Assignments

✧ Bubble Sort

✧ Quick Sort

✧ Minimum Spanning Tree

✧ Tree Traversal

✧ …