What is a “Better” Program?

C++ Object Oriented Programming
Pei-yih Ting
NTOU CS

Goals

-透过一些基本的编码规则，我们可以写出一个“好”一点的C程式
-除了正确性之外，程式短一点?? 執行快一点??
-“好”？（in terms of test, debug, review, and extension）
  1. 容易了解，没有逻辑上不紧密结合的资料变量或是叙述
  2. 自我解释
  3. 和观念上的运作模型一致
  4. 容易修改，不容易改错
  5. 没有容易错误的语法
-正确性无关：以下给你一个很简单的例子，共有7个版本，执行结果都是正确的

Version 1

```c
#include <stdio.h>
void main()
int d[] = {12, 3, 37, 8, 24, 15, 5, 33};
int n = 8;
int *d1, *d2;
int *p;
int *e;

    d1 = d;
    d2 = d+n;
    while (d1<d2)
    {
        p = d1;
        e = d1 + 1;
        while (e<d2)
        {
            if (*e<*p) p = e;
            e++;
        }
        n = *p;
        *p = *d1;
        *d1 = n;
        d1++;
    }
printf("Sorted data:

da1 = d;
d2 = d+n;
while (d1<d2)
{
    p = d1;
e = d1 + 1;
while (e<d2)
{
    if (*e<*p) p = e;
e++;
}
e = *p;
e = *d1;
```
Execution Results

Sorted data:
3 5 8 12 15 24 33 37

由小至大按顺序排列

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

Version 2

01 #include <stdio.h>
02 void main()
03 {
04 int d[] = {12, 3, 37, 8, 24, 15, 5, 33};
05 int n = sizeof(d) / 4;
06 int i, j, k;
07 
08     i = 0;
09     while (i<n)
10         {
11             k = i;
12             j = i + 1;
13             while (j<n)
14                 {
15                     if (d[j]<d[k]) k = j;
16                 }
17             j = d[k];
18             d[k] = d[i];
19             d[i] = j;
20         }
21     printf("Sorted data:\n");
22     i = 0;
23     while (i<n)
24         {
25             printf("%d", d[i]);
26             i = i + 1;
27         }
28     printf("\n");
29     i = i + 1;
30     while (i<n)
31         {
32             printf("%d", d[i]);
33             i = i + 1;
34         }
35 }
Flowchart of the Program

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?

- Are the identifiers used meaningful??
  e.g.
  
  Hw ds J lk te st?
  or
  How does John like the steak?

Version 3

```
#include <stdio.h>

void main() {
    int data[] = {12, 3, 37, 8, 24, 15, 5, 33};
    int ndata = sizeof(data) / sizeof(int);
    int i, j;
    int min;
    int swapTmp;
    i = 0;
    while (i < ndata) {
        int min = i;
        j = i + 1;
        while (j < ndata) {
            if (data[j] < data[min]) min = j;
            j = j + 1;
        }
        swapTmp = data[min];
        data[min] = data[i];
        data[i] = swapTmp;
        i = i + 1;
    }
    printf("Sorted data:\n");
    i = 0;
    while (i < ndata) {
        printf(" %d", data[i]);
        i = i + 1;
    }
    printf("\n");
}
```

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 subarrays, the inner one goes through each subarray 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
  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 usually more semantically constrained repetition construct in C language --- repeat for a predetermined number of times
  for (i=0; i<count; i++)
  {
  ...
  }

Code That Further Illustrates Itself

- **Function** is a powerful construct to **abstract** ideas, not just a utility for saving your typing time.
  --- 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 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     for (i=0; i<ndata; i++)
11     {
12         min = i;
13         for (j=i+1; j<ndata; j++)
14         {
15             if (data[j]<data[min]) min = j;
16         }
17         swapTmp = data[min];
18         data[min] = data[i];
19         data[i] = swapTmp;
20     }
21     printf("Sorted data:\n");
22     for (i=0; i<ndata; i++)
23     printf(" %d", data[i]);
24     printf("\n");
25 }

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     for (i=0; i<ndata; i++)
13     {
14         min = i;
15         for (j=i+1; j<ndata; j++)
16             if (data[j]<data[min]) min = j;
17         swap(&data[i], &data[min]);
18     }
19     printf("Sorted data:
");
20     printArrayContents(data, ndata);
21 }

06 void swap(int *x, int *y)
07 {
08     int tmp;
09     tmp = *x;
10     *x = *y;
11     *y = tmp;
12 }

06 void printArrayContents(int [], int)
07 {
08     for (j=i+1; j<ndata; j++)
09         if (data[j]<data[min]) min = j;
10     swap(&data[i], &data[min]);
11     printf("Sorted data:
");
12     for (i=0; i<ndata; i++)
13         printf(" %d", data[i]);
14     printf("\n");
15 }

07 void printArrayContents(data, ndata)
08 {
09     for (j=i+1; j<ndata; j++)
10         if (data[j]<data[min]) min = j;
11     swap(&data[i], &data[min]);
12     printf("Sorted data:
");
13     for (i=0; i<ndata; i++)
14         printf(" %d", data[i]);
15     printf("\n");
16 }
void printArrayContents(int data[], int ndata) {
    int i;
    printf("Sorted data:\n");
    for (i=0; i<ndata; i++)
        printf(" %d", data[i]);
    printf("\n");
}

void selectionSort(int data[], int ndata) {
    int i;
    findMinimumOfAnArray(&data[i], ndata-i);
}

void findMinimumOfAnArray(int data[], int ndata) {
    int i, min;
    min = 0;
    for (i=1; i<ndata; i++)
    {
        if (data[i]<data[min]) min = i;
    }
    swap(&data[0], &data[min]);
}

void swap(int *x, int *y) {
    int tmp;
    tmp = *x;
    *x = *y;
    *y = tmp;
}

void printArrayContents(int data[], int ndata) {
    int i;
    printf("Sorted data:\n");
    for (i=0; i<ndata; i++)
        printf(" %d", data[i]);
    printf("\n");
}

#include <stdio.h>

void selectionSort(int[], int);
void findMinimumOfAnArray(int[], int);
void swap(int*, int*);
void printArrayContents(int[], int);

void main() {
    int data[] = {12, 3, 37, 8, 24, 15, 5, 33};
    int ndata = sizeof(data) / sizeof(int);
    selectionSort(data, ndata);
    printArrayContents(data, ndata);
}

CODING CONCEPTUAL MODEL

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

In each iteration, just pick the minimum of the subarray and move it to the top.
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 codes 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 which tells 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.

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

Efficiency Issues

- Using expressive name for all identifiers make the program much longer, easy to have typo, 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 will be easier to compose and to maintain.
- 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
- …