CÔNG NGHỆ PHẦN MỀM

Chương 5. LẬP TRÌNH PHẦN MỀM

1

1

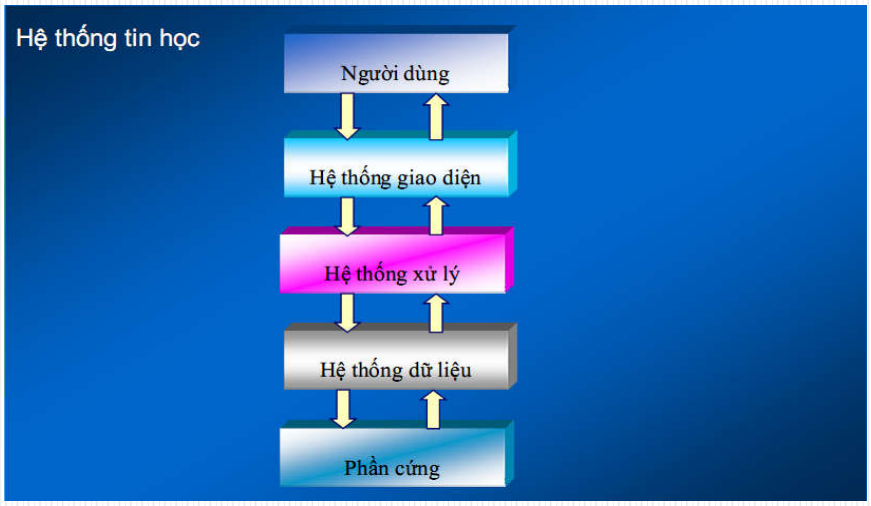
Nội dung

- Kiến trúc của một application
- Mô hình 1 lớp
- Mô hình 2 lớp
- Mô hình 3 lớp

2

2

Mô hình phần mềm

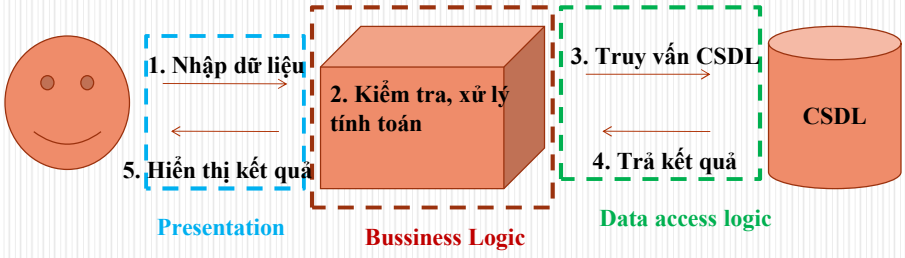


3

3

1. Các kiểu kiến trúc của một application

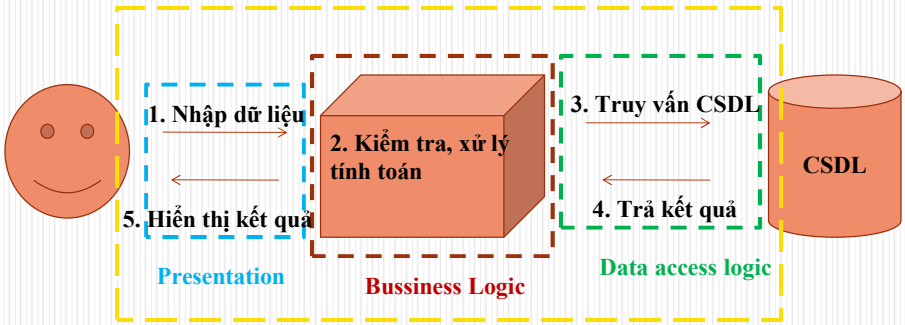
- Quy trình xử lý 1 thao tác thông thường



4

4

Mô hình kiến trúc 1 lớp



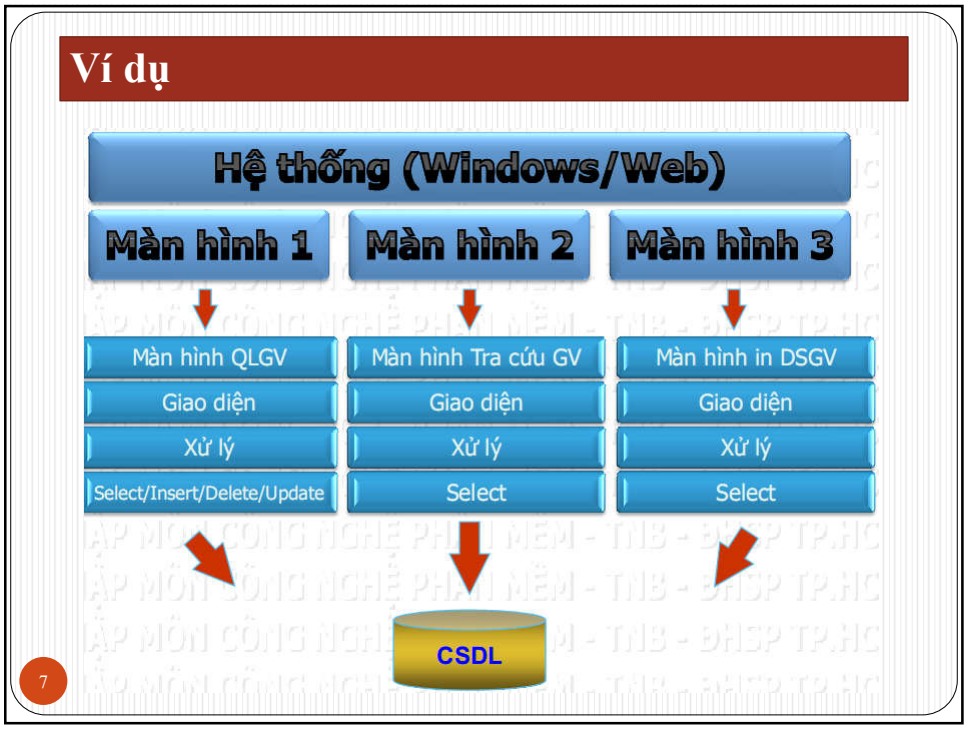
5

5

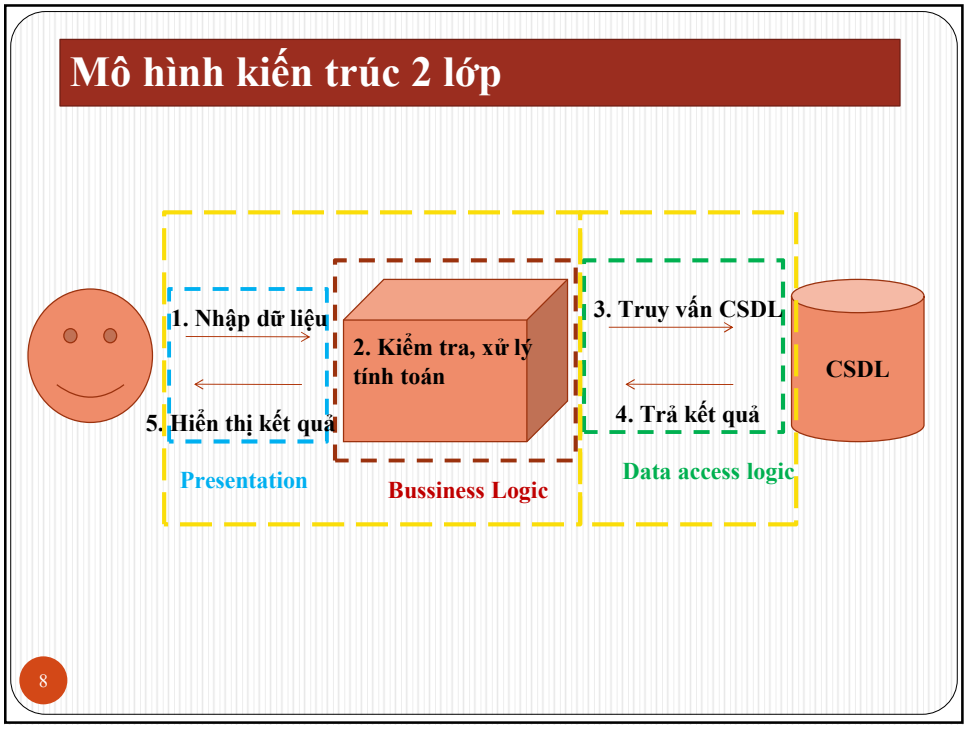
Mô hình kiến trúc 1 lớp

6

6


Ví dụ
Hệ thống (Windows/Web)
Màn hình 1
Màn hình 2
Màn hình 3
Màn hình QLGV
Giao diện
Xử lý
Select/Insert/Delete/Update
Màn hình Tra cứu GV
Giao diện
Xử lý
Select
Màn hình in DSGV
Giao diện
Xử lý
Select
CSDL
7

7


Mô hình kiến trúc 2 lớp
1. Nhập dữ liệu
2. Kiểm tra, xử lý tính toán
3. Truy vấn CSDL
4. Trả kết quả
5. Hiển thị kết quả
CSDL
Presentation
Bussiness Logic
Data access logic
8

8

Mô hình kiến trúc 2 lớp

9

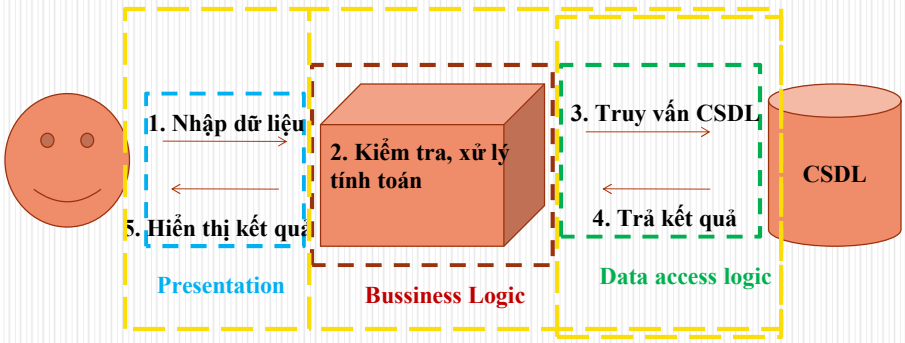
9

Ví dụ

10

10

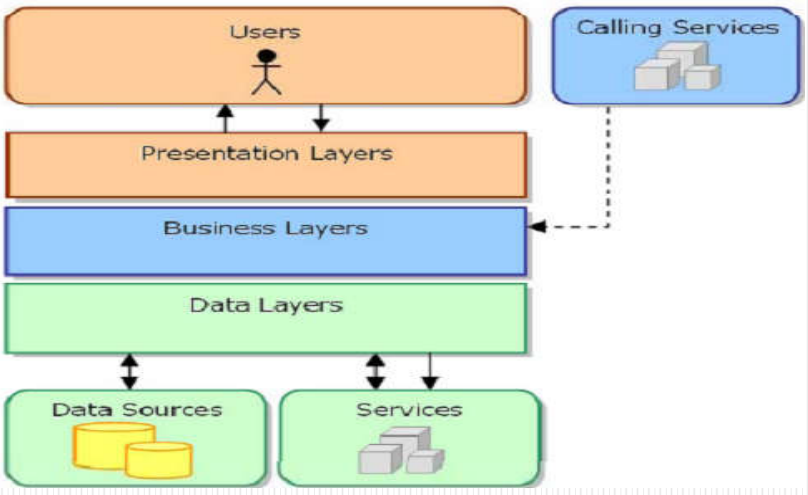
Mô hình kiến trúc 3 lớp



11

11

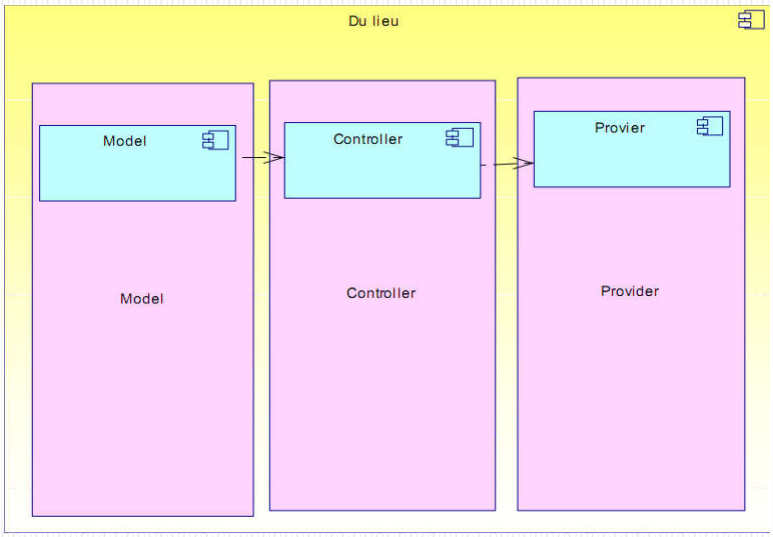
Mô hình 3 tier



12

12

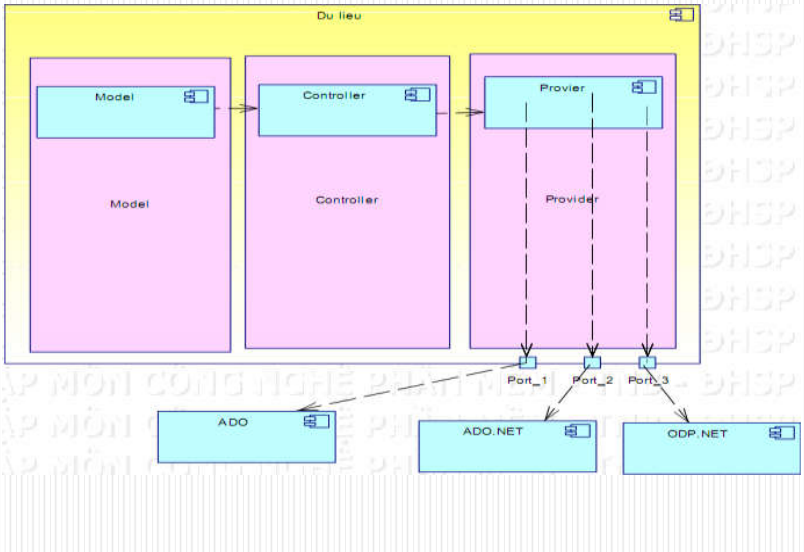
Mô hình 3 tier



13

13

Mô hình 3 tier

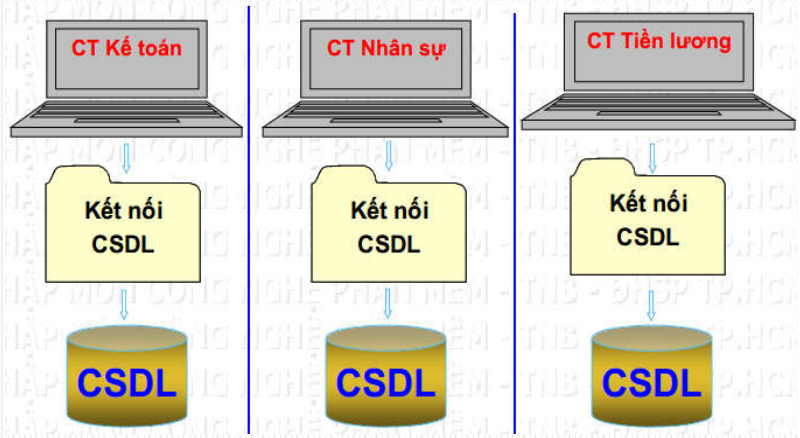


14

14

Mở rộng

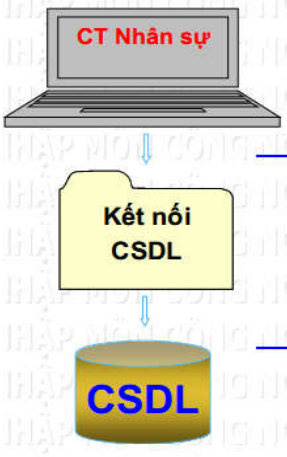
a) CSDL trong ứng dụng quản lý



15

15

Mở rộng



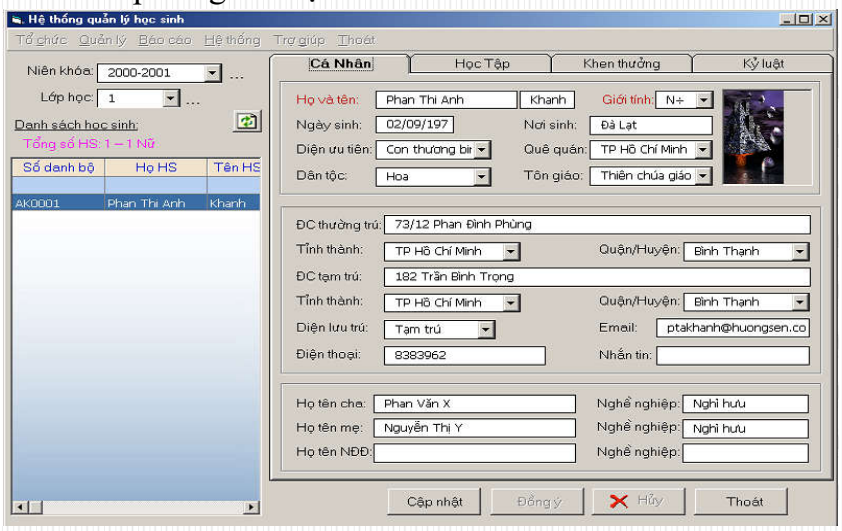
- Thành phần giao diện: Visual Basic, Visual C++, C#, VB.NET, Delphi...
- Giao tiếp dữ liệu: ODBC, DAO, ADODB, ADO.NET,..
- Thành phần dữ liệu: MS Access, SQL Server, Oracle,...

16

16

Ví dụ hệ thống quản lý học sinh

- Thành phần giao diện



17

17

Ví dụ hệ thống quản lý học sinh

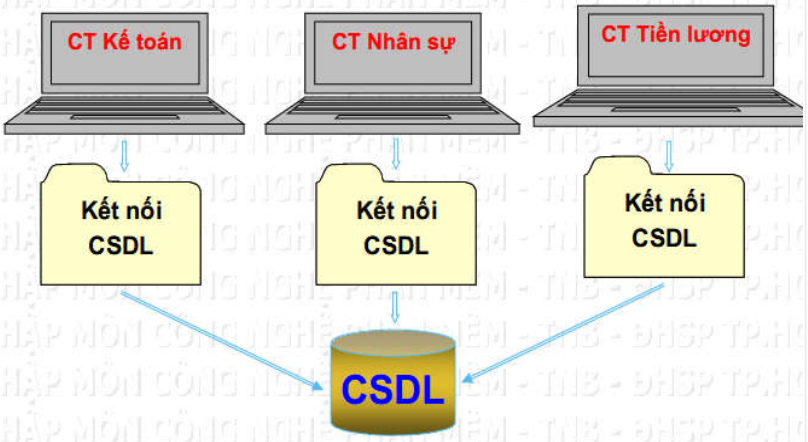
- Thành phần dữ liệu
  - MS Access
  - MS SQL Server
  - Oracle
  - …

18

18

Mở rộng

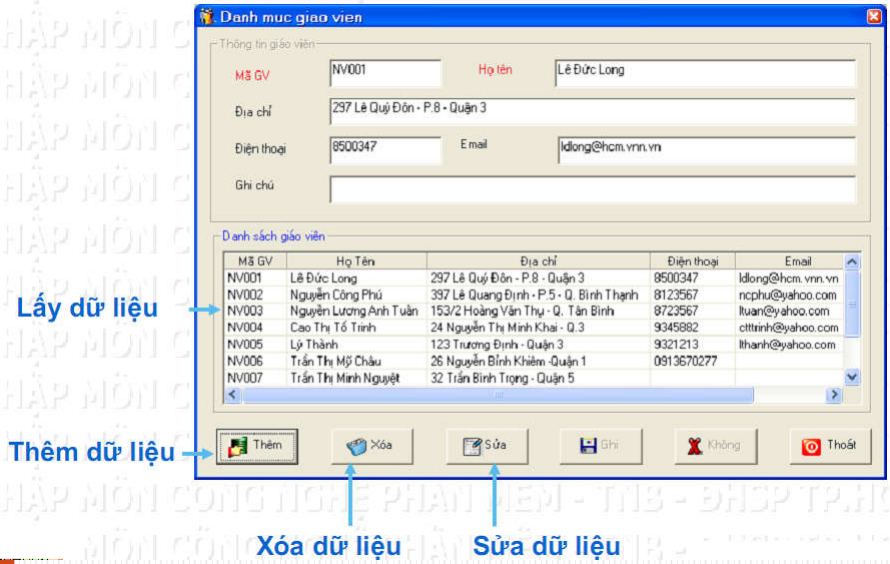
b) CSDL trong ứng dụng quản lý



19

19

Tương tác dữ liệu: thêm/xóa/sửa…



20

20

Thực thi câu lệnh Insert/Delete/Update,…

Thực thi các câu lệnh Insert, Delete, update, Select thông qua đối tượng **Command**

21

21

Một số vấn đề về phong cách lập trình

1. Lựa chọn ngôn ngữ lập trình

- Một trong những lựa chọn công cụ quan trọng nhất dùng tạo ra phần mềm là **ngôn ngữ lập trình**.
- Thông thường việc lựa chọn này được thực hiện mà không có sự phân tích đến chi phí. Kết quả là phần mềm thường có chi phí hơn mức cần thiết.
- Chúng ta nên có kế hoạch tìm hiểu một ngôn ngữ lập trình mới mỗi năm hoặc lâu hơn.

22

22

Một số vấn đề về phong cách lập trình

1. Lựa chọn ngôn ngữ lập trình

- Trong số các NNLT cấp cao, chúng ta nên biết:
  - Ngôn ngữ dạng mệnh lệnh cấp thấp: C
  - Ngôn ngữ dạng mệnh lệnh cấp cao: Ada, Common lisp
  - Ngôn ngữ dạng hàm: Caml, Haskell, Common lisp.
  - Ngôn ngữ hướng đối tượng: C#, Java, Smalltalk…
  - Ngôn ngữ logic: Prolog

23

23

Một số vấn đề về phong cách lập trình

1. Lựa chọn ngôn ngữ lập trình

- Một số ngôn ngữ được ưa thích khác:
  - PHP
  - Javascript or ECMAScript
  - Python
  - Ruby
  - C++
- Xét về mặt hiệu quả, thông thường mọi người thường chọn ngôn ngữ cấp cao để tìm hiểu. Vì các NNLT cấp cao thường sử dụng bộ nhớ quản lý tự động, là yêu cầu của đa số ứng dụng thời gian thực (real – time).

24

24

Một số vấn đề về phong cách lập trình

1. Lựa chọn ngôn ngữ lập trình

- Khi tạo một phần mềm (có ứng dụng quan trọng), chúng ta nên chọn 1 ngôn ngữ với các chuẩn bởi các tổ chức tiêu chuẩn hóa (ISO, ANSI, IEEE, …).
- Một ngôn ngữ được kiểm soát điều khiển bởi một nhà cung cấp (hoặc tệ hơn bởi một cá nhân nào đó). Nó có thể thay đổi theo những cách mà một đầu tư lớn thành vô giá trị sau một đêm. Tuy nhiên những ngôn ngữ loại này thường đem lại lợi ích là thường cung cấp các free, open-source.

25

25

Một số vấn đề về phong cách lập trình

1. Lựa chọn ngôn ngữ lập trình

- Một số ngôn ngữ chuẩn:
  - C
  - C++
  - C#
  - Common Lisp
  - Prolog
  - ECMAScript
  - Ada
- Một số ngôn ngữ không chuẩn:
  - Java
  - Python
  - Ruby

26

26

Một số vấn đề về phong cách lập trình

1. Lựa chọn ngôn ngữ lập trình

- Một số nhà sản xuất (đơn) cung cấp ngôn ngữ đóng gói:
  - VisualBasic (Microsoft)
  - Delphi (Borland)
  - Rebol (RebolTechnologies)

27

27

Một số vấn đề về phong cách lập trình

1. Lựa chọn ngôn ngữ lập trình

- **Maintainable code (Bảo trì code)**

  Để việc bảo trì code được thuận lợi, code cần thỏa những tiêu chuẩn sau:
  - Understandable
  - Extendible
  - Modular
  - Reusable

28

28

Một số vấn đề về phong cách lập trình

1. Lựa chọn ngôn ngữ lập trình

- **Understandable code**

Code được đọc bởi:

- The compiler (trình biên dịch).
  - Code cần tuân thủ theo cú pháp và ngữ nghĩa của NNLT.
- The text editor (chương trình soạn thảo).
  - Code cần thuân thủ thêm những cú pháp riêng.
- Other tools (các công cụ khác).
  - Code phải chịu thêm những ràng buộc khác.
- Maintainers (người bảo trì)

29

29

Một số vấn đề về phong cách lập trình

2. Một số vấn đề trong khi viết chương trình (coding)

a. **Indentation (thụt đầu dòng)**

- Là yếu tố cần thiết để làm code dễ hiểu.
- Nó không thuộc về phong cách mỗi người.
- Nên dùng chuẩn theo qui định.
- Sử dụng một chuẩn cho toàn bộ chương trình.
- Một thụt đầu dòng thường khoảng 2 ký tự trống

30

30

2. Một số vấn đề trong khi viết chương trình (coding)

Cách đặt dấu ngoặc:

```
while (a > b)
  a = f (a);

if (g (x))
  x++;
```

Style 1:

```
void fun (int a) {
  int x;
  ...
}

if (x > y) {
  fun (x);
  ...
}
```

Style 2:

```
void fun (int a)
{
  int x;
  ...
}

if (x > y)
  {
    fun (x);
    ...
  }
```

31

31

2. Một số vấn đề trong khi viết chương trình

b. Spacing (khoảng trống)

- Cũng như thụt đầu dòng, khoảng trống cũng là yếu tố cần thiết để làm code dễ hiểu.
- Nó không thuộc về phong cách mỗi người.
- Nên dùng chuẩn theo qui định.
- Sử dụng một chuẩn cho toàn bộ chương trình.

32

32

2. Một số vấn đề trong khi viết chương trình

b. Spacing (khoảng trống)

Xét cách viết code trên JAVA, C, C++: Đối với các phép toán số học hoặc logic, nên có khoảng trống ở mỗi bên.

```
a=10;
b= 20;
f(a + 3*b);
a>b && c>=10*x
```

Không nên

```
a = 10;
f(a + 3 * b);
a > b && c >= 10 * x
```

Nên

33

33

2. Một số vấn đề trong khi viết chương trình

b. Spacing (khoảng trống)

Xét cách viết code trên JAVA, C, C++: Một dấu , hoặc ; không đứng sau 1 khoảng trống nhưng luôn đứng trước 1 khoảng trống.

```
a=10;
b= 20;
f(a + 3*b);
a>b && c>=10*x
```

Không nên

```
a = 10;
f(a + 3 * b);
a > b && c >= 10 * x
```

Nên

34

34

2. Một số vấn đề trong khi viết chương trình

b. Spacing (khoảng trống)

Xét cách viết code trên JAVA, C, C++: Một dấu , hoặc ; không đứng sau 1 khoảng trống nhưng luôn đứng trước 1 khoảng trống.

```
int x ; float y ;
for (i = 0;i < n;i++)
```

→

```
int x; float y;
for (i = 0; i < n; i++)
```

Không nên **Nên**

35

35

2. Một số vấn đề trong khi viết chương trình

b. Spacing (khoảng trống)

Xét cách viết code trên JAVA, C, C++: Có thể có hoặc không có khoảng trống giữa tên hàm với danh sách hàm.

```
f(x, y, z);
int
g(int x, int y)
```

```
f (x, y, z);
int
g (int x, int y)
```

36

36

2. Một số vấn đề trong khi viết chương trình

b. Spacing (khoảng trống)

Xét cách viết code trên JAVA, C, C++: Nên có khoảng trống giữa các từ khóa *if, while, for* với biểu thức sau nó.

```
for(i = 0; i < n; i++)
while(f(x) > y)
```

Không nên

```
for (i = 0; i < n; i++)
while (f(x) > y)
```

Nên

37

37

2. Một số vấn đề trong khi viết chương trình

b. Spacing (khoảng trống)

Xét cách viết code trên JAVA, C, C++: Không có khoảng trống giữa dấu ngoặc với biểu thức bên trong

```
for(i = 0; i < n; i++)
while(f(x) > y)
```

```
x = tab[y];
```

38

38

2. Một số vấn đề trong khi viết chương trình

c. Cấu trúc lệnh

Xét cách viết code trên JAVA, C, C++: Dùng cấu trúc lệnh rõ ràng nhất.

```
int i = 0;
while (i < n) {
  ...
  i++;
}
```

```
for (int i = 0; i < n; i++) {
  ...
}
```

Không nên **Nên**

39

39

2. Một số vấn đề trong khi viết chương trình

c. Cấu trúc lệnh

Xét cách viết code trên JAVA, C, C++: Dùng cấu trúc lệnh rõ ràng nhất.

```
i = i + 3;

i += 1;
```

```
i += 3;

i++;
```

Không nên **Nên**

40

40

2. Một số vấn đề trong khi viết chương trình

d. Lệnh return

Xét cách viết code trên JAVA, C, C++: return không phải là một hàm nên không cần đặt giá trị vào trong dấu ngoặc.

```
return (a + b * c);
```

```
return a + b * c;
```

Không nên **Nên**

41

41

2. Một số vấn đề trong khi viết chương trình

d. Lệnh return

Khi dùng biểu thức điều kiện, nếu có thể cần tránh lặp lại code.

```
if (a > b)
  return f(a);
else
  return g(b);

if (a > b)
  people[current_person].relatives.next.data[x] = f(a, 2);
else
  people[current_person].relatives.next.data[x] = f(b, 3);
```

Không nên

```
return a > b ? f(a) : g(b);

people[current_person].relatives.next.data[x] =
  a > b ? f(a, 2) : f(b, 3);
```

Nên

42

42

2. Một số vấn đề trong khi viết chương trình

e. Định danh – identifier

- Lớp (class): Thường dùng **danh từ** để đặt tên. Ví dụ: person, vehicle, course,… Vì lớp là đối tượng không thể là hành động, nếu dùng động từ thì sẽ nhầm lẫn với các phương thức.
- Hàm và phương thức dùng để thực hiện một việc nào đó: Tên thường bắt đầu bằng **động từ**.
- Có 2 mẫu để định danh:
  - Viết hoa các ký tự đầu của từ (phương thức: viết thường ký tự đầu tiên).

  Ví dụ: computeNext**I**tem**I**n**L**ist, **F**our**W**heel**V**ehicle.
  - Dùng dấu underscore để phân cách các từ.

  Ví dụ: compute_next_item_in_list, four_wheel_vehicle
- Nên chọn 1 mẫu và tuân thủ trong cả chương trình.

43

43

2. Một số vấn đề trong khi viết chương trình

e. Định danh – identifier

- Không nên đặt tên quá dài hoặc quá ngắn. Tùy trường hợp mà định danh cho phù hợp.

  Ví dụ:
  - Tên dài tốt hơn tên ngắn, chẳng hạn **temperature** rõ ràng hơn **temp** hoặc **t**.
  - Tên ngắn tốt hơn, chẳng hạn **x, y** để chỉ 2 số bất kỳ thay vì **theFirstArbitraryNumber, theSecondArbitraryNumber**.
  - Hoặc **i, j** là biến chạy cho cho for hoặc chỉ số mảng thay vì **arrayInDexes**. Tương tự ta dùng **n** chỉ số thành phần của mảng thay vì **theNumberOfElements.**

44

44

2. Một số vấn đề trong khi viết chương trình

f. Chú thích - comment

- Mục đích của chú thích
  - Để làm rõ nghĩa những đoạn code khó.
  - Để ghi nhận lại tác giả và ngày tạo hoặc chỉnh sửa chương trình.

Ví dụ

```
// it does not matter which element we pick,
// so pick the first one. RS 2007-11-10
element = a[0];
```

45

2. Một số vấn đề trong khi viết chương trình

f. Chú thích - comment

- Chú thích không phải là diễn giải code.

Ví dụ

Bad:

```
while (i < n) { // loop until i is equal to n
  ...
  i++; // increment i
}
```

Good:

```
while (i < n) { // handle every element
  ...
  i++; // we are done with this element, do the next one
}
```

46

2. Một số vấn đề trong khi viết chương trình

f. Chú thích - comment

- Không nên chú thích quá mức cần thiết.

Ví dụ

Bad:

```
// Function: tail
// Author: Robert Strandh
// Version information:
//   Version 1: 2007-11-10
// Name: tail
// Number of arguments: 1
// Arguments descriptions:
//   Argument 1:
//     Name : l
//     Type : list
// Type of return value: A list or NULL.
list
tail(list l)
{
  return l.tail;
}
```

47

47

2. Một số vấn đề trong khi viết chương trình

g. Tránh viết code "xoắn" nhau – avoid convoluted code

Bad:

```
if (a > b)
  return true;
else
  return false;
```

Good:

```
return a > b;
```

Bad:

```
return p.next == NULL ? NULL : p.next;
```

Good:

```
return p.next;
```

48

48

2. Một số vấn đề trong khi viết chương trình

g. Tránh viết code "xoắn" nhau – avoid convoluted code

Bad:

```
int
countNodes(tree t)
{
  if (t.left == NULL)
    if (t.right == NULL)
      return 1;
    else
      return 1 + countNodes(t.right)
  else
    if (t.right == NULL
      return 1 + countNodes(t.left)
    else
      return 1 +
             countNodes(t.left) +
             countNodes(t.right);
}
```

49

49

2. Một số vấn đề trong khi viết chương trình

g. Tránh viết code "xoắn" nhau – avoid convoluted code

Good:

```
int
countNodes(tree t)
{
  return t == NULL ?
         0 :
         1 + countNodes(t.left) + countNodes(t.right);
}
```

50

50

2. Một số vấn đề trong khi viết chương trình

h. Nên dùng hằng (constant) thay vì ghi giá trị trực tiếp

Bad:

```
int *tab = malloc(256 * sizeof(int));

for (i = 0; i < 256; i++)
  tab[i] = 0;
```

Better:

```
#define SIZE 256

int *tab = malloc (SIZE * sizeof(int));

for (i = 0; i < SIZE; i++)
  tab[i] = 0;
```

51

51

2. Một số vấn đề trong khi viết chương trình

i. Nên gán giá trị cho biến ngay khi mới khai báo (nếu có thể)

Bad:

```
int n;
...
n = f (a, b);
```

Good:

```
int n = f (a, b);
```

52

52

2. Một số vấn đề trong khi viết chương trình

j. Giảm phạm vi của biến (nếu có thể)

Bad:

```
int i;
int j;
...
while (i < f (x)) {
  ...
  while (j < g (y)) {
    ...
  }
}
```

Good:

```
int i;
...
while (i < f (x)) {
  int j;
  ...
  while (j < g (y)) {
    ...
  }
}
```

53

53

2. Một số vấn đề trong khi viết chương trình

k. Tránh lặp lại code (code duplication)

- Việc lặp lại code sẽ làm cho quá trình bảo trì khó khăn hơn, vì phải tìm tất cả các đoạn code đó để xem xét và sửa chữa.

l. Nên dùng nhiều hàm và phương thức con (nếu có thể).

Đây là cách hiệu quả để chia nhỏ bài toán, nhằm giúp đơn giản hóa vấn đề và dễ dàng quản lý các đối tượng, phương thức trong chương trình.

54

54

2. Một số vấn đề trong khi viết chương trình

m. Tối ưu trình biên dịch và bộ xử lý

```
for (i = 0; i < n - 1; i++) {
  a[i + 1] = ...;
  b[i + 1] = ...;
}
```

Bad

```
for (i = 1; i < n; i++) {
  a[i] = ...;
  b[i] = ...;
}
```

Good

```
for (i = 0; i < n; i++) { ... }
... // lots of code
for (; i < 2 * n; i++) { ... }
}
```

```
for (i = 0; i < n; i++) { ... }
... // lots of code
for (i = n; i < 2 * n; i++) { ... }
```

55

55

Bài tập

- Hãy tìm trong đoạn code sau những vấn đề mà theo bạn viết chưa hay? Vì sao?
- Bạn hãy chỉnh sửa những vấn đề đó để code dễ hiểu và tối ưu hơn.

Câu 1.

```
void dempt(int a[],int n)
{
        int dpt=0;
        for(int i=0;i<n;i++)
        if(a[i]%2==0)
                dpt+=1;
}
```

Câu 2.

```
void sapxepgiam(int a[],int n)
{
   int t;
        for(int i=0;i<n-1;i++)
                if(a[i]<a[i+1])
                {
                        t=a[i];
                        a[i]=a[i+1];
                        a[i+1]=t;
                }
}
```

56

56

Câu 3.

```
/* Hàm tìm max của mang a và cho biết max đó có là số nguyên tố */

void tim_max(int a[100],int n)
{
        int max=a[0];
        for(int i=0;i<=n-1;i++)
        {
                if(a[i]>max)
                        max=a[i];
        }
                printf("\n so lon nhat la: %d",max);

                        for(int i=2;i<max-1;i++)
                        if(max%i==0)
                        {
                        printf("\n max khong phai la so nguyen to");
                        return;
                        }
                        printf("\n max la so nguyen to");

}
```

57

Câu 4.

```
int InsertNode(Node* &root, Node* p)
 {
  if(root ==NULL)
  {
          root = p;
          return 1
   }
  else
  {
         if(root->key ==p->key)
                return 0;
          if(root->key > p->key)
                return     InsertNode(root->left,
     p);
          if(root->key < p->key)
                return            InsertNode(root-
     >right,p) ;
  }
```

```
int USCLN(int a, int b)
{
   a=abs(a);
   b=abs(b);
   while(a!=b)
   {
      if(a>b)
         a=a-b;
      else
         b=b-a;
   }
   return a;
}
```

```
int TimMax (int a[], int n)
{
      int max, i = 1;

      max = a[0];
      while ( i < n )
      {
            if ( a[i] > max )
                  max = a[i] ;
            i++;
      }
      return max;
}
```

58

```
int normalLength = (int) (1000.0 * 60.0 / (tempo * 2.0));
double lastTick = 0;
double currentTick;
double tickLength = seq.getMicrosecondLength() / seq.getTickLength() / 1000.0;
double events = (seq.getTracks())[0].size();
MidiEvent midiEvent;
MidiMessage midiMessage;
byte[] messageData;
Note note;
TemporalDeviation deltaT;

for (int i = 0 ; i < events; i++) {
    midiEvent = (seq.getTracks())[0].get(i);
    midiMessage = midiEvent.getMessage();
    if (midiMessage.getStatus() == NOTE_ON) {
        currentTick = (double) midiEvent.getTick();
        messageData = midiMessage.getMessage();
        note = new Note((int) (messageData[messageData.length-1] & 0xFF));
        notes.add(note);
        if (lastTick == 0)
            deltaT = new TemporalDeviation(0);
        else
            deltaT = new TemporalDeviation (normalLength -
                                            (int)((currentTick - lastTick)
                                                  * tickLength));
        deviations.add(deltaT);
        lastTick = currentTick;
    }
}
```

59

59

Yêu cầu bài tập nhóm – Đồ án

- Thực hiện khảo sát hiện trạng cho đồ án môn học
- Vẽ mô hình BPM mô tả quy trình nghiệp vụ
- Vẽ mô hình dữ liệu mức quan niệm CDM
- Vẽ mô hình phân cấp chức năng BFD
- Vẽ mô hình luồng dữ liệu DFD
- Thiết kế dữ liệu, xử lý và giao diện
- **Lập trình phần mềm**

60

60

61