Trịnh Thành Trung (ThS)
trungtt@soict.hust.edu.vn

Bài 5
THIẾT KẾ CHƯƠNG TRÌNH

Nội dung

1. Nguyên tắc chung
2. Thiết kế giải thuật
3. Thiết kế dữ liệu

1.
Nguyên tắc chung

Trong thiết kế chương trình

Phẩm chất của chương trình tốt

- Phẩm chất của 1 chương trình tốt
  - *Cấu trúc tốt*
  - *Logic chương trình + các biểu thức được diễn đạt theo cách thông thường*
  - *Tên dùng trong chương trình có tính chất miêu tả*
  - *Chú thích hợp lý*
  - *Tôn trọng chiến lược divide/conquer/association*
- Làm thế nào để tạo ra chương trình có phẩm chất tốt
  - *Thiết kế top-down*
  - *Tinh chỉnh từng bước*

Nguyên tắc chung

Đơn giản

- Thể hiện giải thuật như nó vốn có, đừng quá kỳ bí
- Lựa chọn cấu trúc dữ liệu sao cho việc viết giải thuật bằng NNLT cụ thể là đơn giản nhất
- Tìm cách đơn giản hóa các biểu thức
- Thay những biểu thức lặp đi lặp lại bằng CT con tương ứng

Nguyên tắc chung

Trực tiếp

- Sử dụng thư viện mọi lúc có thể
- Tránh việc kiểm tra điều kiện không cần thiết

Rõ ràng

- Dùng các cặp dấu đánh dấu khối lệnh để tránh nhập nhằng
- Đặt tên biến, hàm, .. sao cho tránh được nhầm lẫn
- Không chắp vá các đoạn mã khó hiểu mà nên viết lại

Nguyên tắc chung

Cấu trúc tốt

- Tôn trọng tính cấu trúc của chương trình theo từng mô thức lập trình:
  - *Module: hàm/ thủ tục*
  - *Hướng đối tượng: lớp*
  - *Hướng thành phần: thành phần*
  - *Hướng dịch vụ: dịch vụ*
- Viết và kiểm thử dựa trên cấu trúc phân cấp của chương trình
- Tránh hoàn toàn việc dùng *goto*

→ Nếu cần thì nên viết giải thuật bằng giả ngữ, rồi mới viết bằng 1 NNLT cụ thể

2.
Thiết kế giải thuật

Algorithms

Thiết kế giải thuật

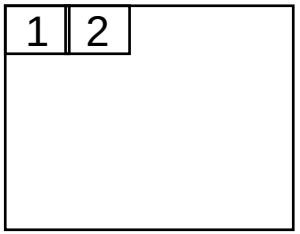
- Chia bài toán ra thành nhiều bài toán nhỏ hơn
- Tìm giải pháp cho từng bài toán nhỏ
- Gộp các giải pháp cho các bài toán nhỏ thành giải pháp tổng thể cho bài toán ban đầu

→ Đơn giản hóa bài toán bằng cách trừu tượng hóa: làm cái gì thay vì làm như thế nào

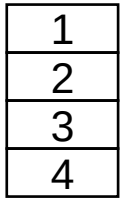
  - *Ví dụ: các hàm ở mức trừu tượng*
    - Hàm sắp xếp 1 mảng các số nguyên
    - Hàm nhập vào / xuất ra các ký tự: getchar() , putchar()
    - Hàm toán học : sin(x), sqrt(x)

Thiết kế từ dưới lên
Bottom-up design

- Bottom-up design ☹
  - *Thiết kế chi tiết 1 phần*
  - *Thiết kế chi tiết 1 phần khác*
  - *Lặp lại cho đến hết*
- Bottom-up design in programming
  - *Viết phần đầu tiên của CT 1 cách chi tiết cho đến hết*
  - *Viết phần tiếp theo của CT 1 cách chi tiết cho đến hết*
  - *Lặp lại cho đến hết*



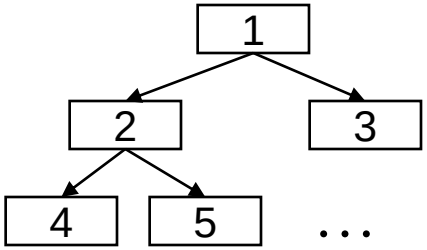
...



...

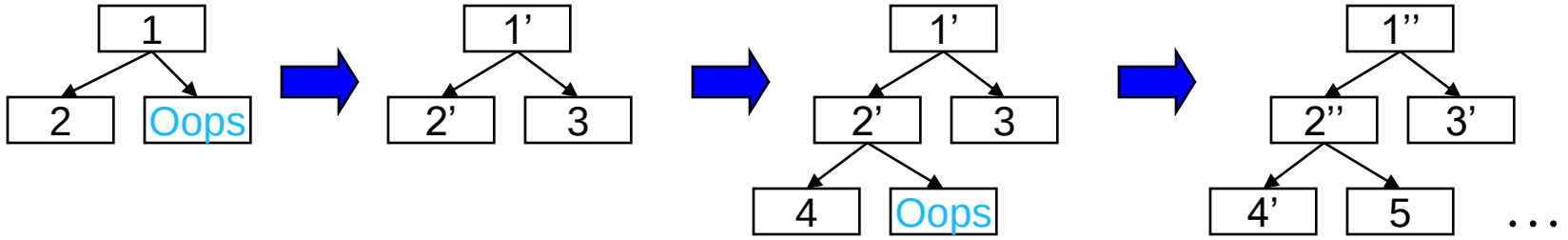
Thiết kế từ trên xuống
Top-down design

- Top-down design ☺
  - *Thiết kế toàn bộ sản phẩm một cách sơ bộ, tổng thể*
  - *Tinh chỉnh cho đến khi hoàn thiện*
- Top-down design in **programming**
  - *Phác họa hàm main() (bằng các lệnh giả ngữ - pseudocode)*
  - *Tinh chỉnh từng lệnh giả ngữ*
    - Công việc đơn giản => thay bằng real code
    - Công việc phức tạp => thay bằng lời gọi hàm
  - *Lặp lại sâu hơn, cụ thể, chi tiết hơn*
  - *Kết quả: Sản phẩm có cấu trúc phân cấp tự nhiên*



Thiết kế trên xuống
trong thực tiễn

- Định nghĩa hàm main() bằng giả ngữ
- Tinh chỉnh từng lệnh giả ngữ
  - *Nếu gặp sự cố: xem lại thiết kế, và…*
  - *Quay lại để tinh chỉnh giả ngữ đã có, và tiếp tục*
- Lặp lại (trong hầu hết các trường hợp) ở mức sâu hơn, cụ thể hơn, cho đến khi các hàm được định nghĩa xong



Ví dụ Text Format

- Mục tiêu :
  - *Minh họa good program và programming style*
    - Đặc biệt là module hóa mức hàm và top-down design
  - *Minh họa cách đi từ vấn đề đến viết code*
    - Ôn lại và mô tả cách xây dưng chương trình C
- Text formatting
  - *Đầu vào: ASCII text, với hàng loạt dấu cách và phân dòng*
  - *Đầu ra: Cùng nội dung, nhưng căn trái và căn phải*
    - Dồn các từ tối đa có thể trên 1 dòng 50 ký tự
    - Thêm các dấu cách cần thiết giữa các từ để căn phải
    - Không cần căn phải dòng cuối cùng
  - *Để đơn giản hóa, giả định rằng :*
    - 1 từ kết thúc bằng dấu cách space, tab, newline, hoặc end-of-file
    - Không có từ nào quá 20 ký tự

Input và Output

INPUT

```
Tune   every   heart    and    every      voice.
                    Bid every      bank withdrawal.
Let's all    with our    accounts  rejoice.
In funding  Old  Nassau.
        In funding Old Nassau we spend more money every year.
Our banks      shall give, while      we     shall    live.
We're funding                      Old Nassau.
```

OUTPUT

```
Tune every heart and every voice. Bid  every  bank
withdrawal. Let's all with our  accounts  rejoice.
In funding Old Nassau. In funding  Old  Nassau  we
spend more money every year. Our banks shall give,
while we shall live. We're funding Old Nassau.
```

Phân tích bài toán

- Khái niêm “từ”
  - *Chuỗi các ký tự không có khoảng trắng, tab xuống dòng, hoặc EOF*
  - *Tất cả các ký tự trong 1 từ phải đc in trên cùng 1 dòng*
- Làm sao để đọc và in đc các từ
  - *Đọc các ký tự từ stdin cho đến khi gặp space, tab, newline, or EOF*
  - *In các ký tự ra stdout tiếp theo bởi các dấu space(s) or newline*
- Nếu đầu vào lộn xộn thì thế nào?
  - *Cần loại bỏ các dấu spaces thừa, các dấu tabs, và newlines từ input*

Phân tích
bài toán

- Làm sao có thể căn trái - phải ?
  - *Ta không biết được số dấu spaces cần thiết cho đến khi đọc hết các từ*
  - *Cần phải lưu lại các từ cho đến khi có thể in được trọn vẹn 1 dòng*
- Cần thêm bao nhiêu space vào giữa các từ?
  - *Cần ít nhất 1 dấu space giữa các từ riêng biệt trên 1 dòng*
  - *Có thể thêm 1 vài dấu spaces để phủ kín 1 dòng*

Viết chương trình

- Các cấu trúc dữ liệu chính
  - *Từ - Word*
  - *Dòng - Line*
- Các bước tiếp theo
  - *Viết pseudocode cho hàm main()*
  - *Tinh chỉnh*
- Lưu ý :
  - *Chú thích hàm và một số dòng trống được bỏ qua vì những hạn chế không gian*
    - Phải tôn trọng các quy tắc trình bày mã nguồn khi viết CT thực tế
  - *Trình tự thiết kế là lý tưởng*
    - Trong thực tế, nhiều backtracking sẽ xảy ra

Mức đỉnh

```
int main(void) {
    <Xóa dòng>
    for (;;) {
        <Đọc 1 từ>
        if (<Hết từ>) {
            <In dòng không cần căn phải>
            return 0;
        }
        if (<Từ không vừa dòng hiện tại>) {
            <In dòng có căn lề phải>
            <Xóa dòng>
        }
        <Thêm từ vào dòng>
    }
    return 0;
}
```

Tinh chỉnh Đọc một từ

- Khá phức tạp nên cần tách thành 1 hàm riêng

```
#include <stdio.h>
enum {MAX_WORD_LEN = 20};
int main(void) {
   char word[MAX_WORD_LEN + 1];
   int wordLen;
   < Xóa dòng >
   for (;;) {
      wordLen = ReadWord(word);
      if (<Hết từ>) {
         < In dòng không cần căn phải >
         return 0;
      }
      if (< Từ không vừa dòng hiện tại >) {
         < In dòng có căn lề phải >
         < Xóa dòng >
      }
      < Thêm từ vào dòng >
   }
   return 0;
}
```

```
int ReadWord(char *word) {
   <Bỏ qua whitespace>
   <Lưu các ký tự cho đến MAX_WORD_LEN của từ>
   <Trả về độ dài từ>
}
```

EOF
End-of-File Character

- Các files không kết thúc bằng “EOF character”, vì không tồn tại ký tự đó
- EOF là:
  - *Một giá trị đặc biệt (int) được hàm getchar() hoặc các hàm liên quan trả về để chỉ ra 1 lỗi vào ra*
  - *Được định nghĩa trong stdio.h (thường với giá trị -1)*
  - *Trong môi trường windows, có thể tương đương với mã ASCII của cụm phím tắt Ctl + Z*
  - *Trong môi trường unix, có thể tương đương với mã ASCII của cụm phím tắt Ctl+D*

Sử dụng EOF

- Correct code

```
int c;
c = getchar();
while (c != EOF) {
    …
    c = getchar();
}
```

- Equivalent idiom

```
int c;
while ((c = getchar()) != EOF) {
    …
}
```

- Incorrect code

<?>

```
char c;
while ((c = getchar()) != EOF) {
    …
}
```

Tinh chỉnh
Đọc một từ

```
int ReadWord(char *word) {
   int ch, pos = 0;

   /* Bỏ qua whitespace. */
   ch = getchar();
   while ((ch == ' ') || (ch == '\n') || (ch == '\t'))
      ch = getchar();

   /* Lưu các ký tự vào từ cho đến MAX_WORD_LEN . */
   while ((ch != ' ') && (ch != '\n') && (ch != '\t') && (ch != EOF)) {
      if (pos < MAX_WORD_LEN) {
         word[pos] = (char)ch;
         pos++;
      }
      ch = getchar();
   }
   word[pos] = '\0';

   /* Trả về độ dài từ. */
   return pos;
}
```

Tinh chỉnh Đọc một từ

- ReadWord() chứa 1 vài đoạn code lặp lại => tách thành 1 hàm riêng

`IsWhitespace(ch)`

```
int ReadWord(char *word) {
   int ch, pos = 0;

   /* Bỏ qua whitespace. */
   ch = getchar();
   while (IsWhitespace(ch))
      ch = getchar();

   /* Lưu các ký tự vào từ cho đến MAX_WORD_LEN */
   while (!IsWhitespace(ch) && (ch != EOF)) {
      if (pos < MAX_WORD_LEN) {
         word[pos] = (char)ch;
         pos++;
      }
      ch = getchar();
   }
   word[pos] = '\0';

   /* trả về độ dài từ. */
   return pos;
}
```

```
int IsWhitespace(int ch) {
   return (ch == ' ') || (ch == '\n') || (ch == '\t');
}
```

Tinh chỉnh Lưu từ

- Quay lại main()

<Thêm từ vào dòng> có nghĩa là gì ?

- Tạo 1 hàm riêng cho việc đó

AddWord(word, line, &lineLen)

```
#include <stdio.h>
#include <string.h>
enum {MAX_WORD_LEN = 20};
enum {MAX_LINE_LEN = 50};
int main(void) {
   char word[MAX_WORD_LEN + 1];
   int wordLen;
   char line[MAX_LINE_LEN + 1];
   int lineLen = 0;
   <Xóa dòng>
   for (;;) {
      wordLen = ReadWord(word);
      if (<Hết từ>) {
         <In dòng không căn lề>
         return 0;
      }
      if (<Từ không vừa dòng) {
         < In dòng có căn lề >
         <Xóa dòng>
      }
      AddWord(word, line, &lineLen);
   }
   return 0;
}
```

```
void AddWord(const char *word, char *line, int *lineLen) {
   <Nếu dòng đã chứa 1 số từ, thêm 1 dấu trắng>
   strcat(line, word);
   (*lineLen) += strlen(word);
}
```

Tinh chỉnh
Lưu từ

```
void AddWord(const char *word, char *line, int *lineLen) {

   /* Nếu dòng đã chứa 1 số từ, thêm 1 dấu trắng. */
   if (*lineLen > 0) {
      line[*lineLen] = ' ';
      line[*lineLen + 1] = '\0';
      (*lineLen)++;
   }

   strcat(line, word);
   (*lineLen) += strlen(word);
}
```

Tinh chỉnh In dòng cuối

- <Hết từ> và <In dòng không căn lề> nghĩa là gì?
- Tạo các hàm để thực hiện

```
int main(void) {
   char word[MAX_WORD_LEN + 1];
   int wordLen;
   char line[MAX_LINE_LEN + 1];
   int lineLen = 0;
   <Xóa dòng>
   for (;;) {
      wordLen = ReadWord(word);

      /* Nếu hết từ, in dòng không căn lề */
      if ((wordLen == 0) && (lineLen > 0)) {
         puts(line);
         return 0;
      }
      if (<Từ không vừa dòng>) {
         <In dòng có căn lề>
         <Xóa dòng>
      }
      AddWord(word, line, &lineLen);
   }
   return 0;
}
```

Tinh chỉnh Quyết định in

- <Từ không vừa dòng> Nghĩa là gì?

```
int main(void) {
   char word[MAX_WORD_LEN + 1];
   int wordLen;
   char line[MAX_LINE_LEN + 1];
   int lineLen = 0;
   <Xóa dòng>
   for (;;) {
      wordLen = ReadWord(word);
      /* If no more words, print line
         with no justification. */
      if ((wordLen == 0) && (lineLen > 0)) {
         puts(line);
         return 0;
      }
      /* Nếu từ không vừa dòng, thì … */
      if ((wordLen + 1 + lineLen) > MAX_LINE_LEN) {
         <In dòng có căn lề>
         < Xóa dòng >
      }
      AddWord(word, line, &lineLen);
   }
   return 0;
}
```

Tinh chỉnh
In dòng có căn lề

- <In dòng có căn lề> nghĩa là gì?
- Cần biết trong dòng hiện tại có bao nhiêu từ. Vì vậy ta thêm **numWords** vào hàm main

```
…
int main(void) {
   …
   int numWords = 0;
   <Xóa dòng>
   for (;;) {
      …
      /* Nếu từ không vừa dòng, thì… */
      if ((wordLen + 1 + lineLen) > MAX_LINE_LEN) {
         WriteLine(line, lineLen, numWords);
         <Xóa dòng>
      }

      AddWord(word, line, &lineLen);
      numWords++;
   }
   return 0;
}
```

Tinh chỉnh
In dòng có căn lề

- Hoàn thiện WriteLine()

```
void WriteLine(const char *line, int lineLen, int numWords) {

  <Tính số khoảng trống dư thừa cho dòng>

  for (i = 0; i < lineLen; i++) {
    if (<line[i] is not a space>)
      <Print the character>
    else {
      <Tính số khoảng trống cần bù thêm>

      <In 1 space, cộng thêm các spaces cần bù>

      <Giảm thêm không gian và đếm số từ>
    }
  }
}
```

Tinh chỉnh
In dòng có căn lề

```
void WriteLine(const char *line, int lineLen, int numWords) {
   int extraSpaces, spacesToInsert, i, j;
   /* Tính số khoảng trống dư thừa cho dòng. */
   extraSpaces = MAX_LINE_LEN - lineLen;
   for (i = 0; i < lineLen; i++) {
      if (line[i] != ' ')
         putchar(line[i]);
      else {
         /* Tính số khoảng trống cần thêm. */
         spacesToInsert = extraSpaces / (numWords - 1);

         /* In 1 space, cộng thêm các spaces phụ. */
         for (j = 1; j <= spacesToInsert + 1; j++)
            putchar(' ');

         /* Giảm bớt spaces và đếm từ. */
         extraSpaces -= spacesToInsert;
         numWords--;
      }
   }
   putchar('\n');
}
```

Số lượng các khoảng trống

Ví dụ:
Nếu extraSpaces = 10
và numWords = 5,
thì space bù sẽ là
2, 2, 3, and 3 tương ứng

Tinh chỉnh
Xóa dòng

- <Xóa dòng> nghĩa là gì?
- Tuy đơn giản, nhưng ta cũng viết thành 1 hàm

```
…
int main(void) {
   …
   int numWords = 0;
   ClearLine(line, &lineLen, &numWords);
   for (;;) {
      …
      /* If word doesn't fit on this line, then… */
      if ((wordLen + 1 + lineLen) > MAX_LINE_LEN) {
         WriteLine(line, lineLen, numWords);
         ClearLine(line, &lineLen, &numWords);
      }

      addWord(word, line, &lineLen);
      numWords++;
   }
   return 0;
}
```

```
void ClearLine(char *line, int *lineLen, int *numWords) {
   line[0] = '\0';
   *lineLen = 0;
   *numWords = 0;
}
```

Mô-đun hóa
Modularity

- Với người sử dụng
  - *Input: Văn bản với định dạng lộn xộn*
  - *Output: Cùng nội dung, nhưng trình bày căn lề trái, phải, rõ ràng, sáng sủa*
- Giữa các phần của chương trình
  - *Các hàm xử lý từ : Word-handling functions*
  - *Các hàm xử lý dòng : Line-handling functions*
  - *main() function*

Ưu điểm của mô-đun hóa

- Đọc code: dễ ràng, qua cac mẩu nhỏ, riêng biệt
- Testing : Test từng hàm riêng biệt
- Tăng tốc độ: Chỉ tập trung vào các phần tốc độ còn chậm
- Mở rộng: Chỉ thay đổi các phần liên quan

```
#include <stdio.h>
#include <string.h>

enum {MAX_WORD_LEN = 20};
enum {MAX_LINE_LEN = 50};

int IsWhitespace(int ch) {

/* Return 1 (TRUE) iff ch is a whitespace character. */

   return (ch == ' ') || (ch == '\n') || (ch == '\t');
}
```

```
int ReadWord(char *word) {

/* Read a word from stdin.  Assign it to word.  Return the length
   of the word, or 0 if no word could be read. */

   int ch, pos = 0;

   /* Skip over whitespace. */
   ch = getchar();
   while (IsWhitespace(ch))
      ch = getchar();

   /* Store chars up to MAX_WORD_LEN in word. */
   while (!IsWhitespace(ch) && (ch != EOF)) {
      if (pos < MAX_WORD_LEN) {
         word[pos] = (char)ch;
         pos++;
      }
      ch = getchar();
   }
   word[pos] = '\0';

   /* Return length of word. */
   return pos;
}
```

```
void ClearLine(char *line, int *lineLen, int *numWords) {

/* Clear the given line.  That is, clear line, and set *lineLen
   and *numWords to 0. */

   line[0] = '\0';
   *lineLen = 0;
   *numWords = 0;
}

void AddWord(const char *word, char *line, int *lineLen) {

/* Append word to line, making sure that the words within line are
   separated with spaces.  Update *lineLen to indicate the
   new line length. */

   /* If line already contains some words, append a space. */
   if (*lineLen > 0) {
      line[*lineLen] = ' ';
      line[*lineLen + 1] = '\0';
      (*lineLen)++;
   }
   strcat(line, word);
   (*lineLen) += strlen(word);
}
```

```
void WriteLine(const char *line, int lineLen, int numWords) {

/* Write line to stdout, in right justified form.
   lineLen indicates the number of characters in line.
   numWords indicates the number of words in line. */

   int extraSpaces, spacesToInsert, i, j;

   /* Compute number of excess spaces for line. */
   extraSpaces = MAX_LINE_LEN - lineLen;

   for (i = 0; i < lineLen; i++) {
      if (line[i] != ' ')
         putchar(line[i]);
      else {
         /* Compute additional spaces to insert. */
         spacesToInsert = extraSpaces / (numWords - 1);

         /* Print a space, plus additional spaces. */
         for (j = 1; j <= spacesToInsert + 1; j++)
            putchar(' ');

         /* Decrease extra spaces and word count. */
         extraSpaces -= spacesToInsert;
         numWords--;
      }
   }
   putchar('\n');
}
```

```
int main(void) {

/* Read words from stdin, and write the words in justified format
   to stdout. */

/* Simplifying assumptions:
   Each word ends with a space, tab, newline, or end-of-file.
   No word is longer than MAX_WORD_LEN characters. */

   char word[MAX_WORD_LEN + 1];
   int wordLen;

   char line[MAX_LINE_LEN + 1];
   int lineLen = 0;
   int numWords = 0;

   ClearLine(line, &lineLen, &numWords);

   …
```

```
   …

   for (;;) {
      wordLen = ReadWord(word);

      /* If no more words, print line
         with no justification. */
      if ((wordLen == 0) && (lineLen > 0)) {
         puts(line);
         break;
      }

      /* If word doesn't fit on this line, then... */
      if ((wordLen + 1 + lineLen) > MAX_LINE_LEN) {
         WriteLine(line, lineLen, numWords);
         ClearLine(line, &lineLen, &numWords);
      }

      AddWord(word, line, &lineLen);
      numWords++;
   }
   return 0;
}
```

3.
Thiết kế dữ liệu

Data structure

Thiết kế
dữ liệu

- Cần thiết kế cấu trúc dữ liệu cho phép thực hiện các thao tác sau:
  - *Create: Tạo mới các bộ dữ liệu*
  - *Add: Thêm mới các dữ liệu thành phần*
  - *Search: Tìm kiếm các dữ liệu thành phần*
  - *Free: Hủy cấu trúc dữ liệu*

Thiết kế dữ liệu

- Ví dụ: Bài toán cho các bộ dữ liệu mẫu như sau:
  - *(tên sinh viên, điểm)*
    - (“john smith”, 84)
    - (“jane doe”, 93)
    - (“bill clinton”, 81)
    - …
  - *(tên cầu thủ, vị trí chơi trên sân)*
    - (“Ruth”, 3)
    - (“Gehrig”, 4)
    - (“Mantle”, 7)
    - …
  - *(tên biến, giá trị)*
    - (“maxLength”, 2000)
    - (“i”, 7)
    - (“j”, -10)
    - …

Thanks!

Any questions?

Email me at trungtt@soict.hust.edu.vn