Graphical user interface, text

Description automatically generatedLý giải hết các code

**dwSize**  
Một cấu trúc [**COORD**](https://learn.microsoft.com/en-us/windows/console/coord-str) chứa kích thước của bộ đệm màn hình console, trong các cột và hàng ký tự.

**dwCursorPosition**  
Một cấu trúc [**COORD**](https://learn.microsoft.com/en-us/windows/console/coord-str) chứa tọa độ cột và hàng của con trỏ trong bộ đệm màn hình điều khiển.

**wAttributes**  
Các thuộc tính của các ký tự được ghi vào bộ đệm màn hình bởi hàm [**WriteFile**](https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-writefile) và [**WriteConsole**](https://learn.microsoft.com/en-us/windows/console/writeconsole) , hoặc được lặp lại vào bộ đệm màn hình bởi hàm [**ReadFile**](https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-readfile) và [**ReadConsole**](https://learn.microsoft.com/en-us/windows/console/readconsole) . Để biết thêm thông tin, hãy xem [Thuộc tính ký tự](https://learn.microsoft.com/en-us/windows/console/console-screen-buffers#character-attributes) .

**srWindow**  
Một cấu trúc [**SMALL\_RECT**](https://learn.microsoft.com/en-us/windows/console/small-rect-str) có chứa tọa độ bộ đệm màn hình điều khiển của góc trên bên trái và góc dưới bên phải của cửa sổ hiển thị.

**dwMaximumWindowSize**  
Một cấu trúc [**COORD**](https://learn.microsoft.com/en-us/windows/console/coord-str) chứa kích thước tối đa của cửa sổ bảng điều khiển, trong các cột và hàng ký tự, với kích thước bộ đệm màn hình và phông chữ hiện tại và kích thước màn hình.

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

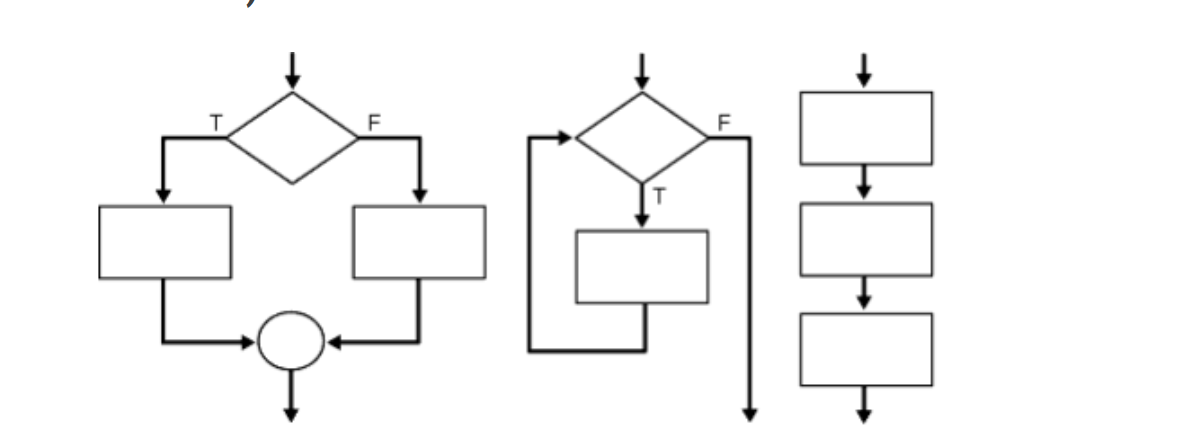
Text

Description automatically generated

Một số hàm mở rộng( hàm tọa độ giải thích sao có cơ chế ấy) mọi người xem thêm: <https://blogkhanhtoan.wordpress.com/2016/03/07/mot-so-ham-mo-rong-trong-cc/>

* Bổ sung

Câu lệnh điều khiển(flow controls):



Trong ngôn ngữ lập trình nói chung, **các câu lệnh (*Statements*)** là tập các chỉ thị mà bản thân ta mong muốn chương trình sẽ thực hiện, ví dụ như việc khai báo một biến (*Variables Declaration*) hay việc sử dụng các toán tử để tính toán,… Các câu lệnh luôn được kết thúc bằng dấu ; và sẽ được trình biên dịch (*compiler*) thực thi ngay tại vị trí nó được khởi tạo.

Dĩ nhiên, khi đối mặt với một vấn đề phức tạp, rắc rối và lắc léo hơn có thể sẽ đòi hỏi ta cần thực lặp lại một đoạn code nhỏ nào đó, hoặc đưa ra các chỉ thị riêng biệt cho từng trường hợp có thể xảy ra trong vấn đề đó,… Vì vậy mà trong C++ đã cung cấp các **Câu lệnh điều khiển (*Flow Controls*)** để thỏa mãn những nhu cầu nói trên, nó sẽ cho phép ta kiểm soát được việc chương trình của mình phải được thực hiện như thế nào, trong trường hợp ra làm sao,…

**Các câu lệnh điều khiển trong C++ được chia làm 3 loại:**

* **Lệnh lựa chọn (*Selection Statements)*:**cho phép ta thể hiện các điều kiện cần để giải quyết bài toán thông qua các phép toán quan hệ, ở đây mình sẽ quan tâm đến các **câu lệnh điều kiện (*Conditional Statements*)**
* **Lệnh lặp (*Iteration Statements*):**cho phép ta thể hiện sự lặp đi lặp lại của một công việc trong chương trình, ở đây mình sẽ quan tâm đến các câu lệnh về **vòng lặp (*Loops*)**
* **Lệnh “nhảy” (*Jump Statements*):**cho phép ta thể hiện sự thay đổi của một câu lệnh điều khiển, chẳng hạn như dừng việc lặp lại dù lệnh lặp vẫn còn có thể lặp tiếp được, ở đây mình sẽ quan tâm đến hai **từ khóa break và continue*(break & continue Statement)***
* **a. Câu lệnh if (if Statement)**
* Đây là cú pháp cơ bản nhất của câu lệnh rẽ nhánh trong C++. Nó cho phép truyền vào một hoặc nhiều biểu thức logic khác nhau bên trong dấu () và sau đó kiểm tra tính đúng sai của chúng, nếu tất cả các điều kiện bên trong nó **đều thỏa mãn**(true), khối lệnh bên trong if sẽ được thực thi.

Diagram

Description automatically generated

* Khi trình biên dịch gặp lệnh if, nó sẽ bắt đầu đánh giá xem điều kiện bên trong sẽ cho giá trị 1 (true) hay 0(false). Nếu true nó sẽ tiếp tục biên dịch các câu lệnh nằm bên trong if, ngược lại nó sẽ bỏ qua và tiếp tục biên dịch các dòng lệnh bên dưới dấu }.

## b. Câu lệnh if…else (if…else Statement)

Lấy lại cái ví dụ trời mưa ban nãy nha, bây giờ nếu ta diễn giải vụ trời mưa này lên chương trình chỉ bằng câu lệnh if đơn thuần, tức ta sẽ chỉ có thể đặt đúng một điều kiện cho nó thôi! Nhưng mà mình lớn rồi, phải lo xa hơn tí chứ đúng không :3 Lỡ giờ trời không mưa thì làm sao? Các bạn có thể nghĩ đến việc viết thêm một dòng lệnh if nữa, nhưng việc làm đó có vẻ như sẽ làm chương trình của tụi mình hơi rối và thừa thải…

Chính vì lẽ đó, C++ đã mở rộng câu lệnh if ra, cho ta thêm một câu lệnh mới đó là **if…else** (nhà sáng lập tâm lý ghê chưa :3).

* Sự khác biệt rõ ràng nhất ở đây là có thêm sự hiện diện của khối else nữa đúng không các bạn :3 Thật vậy, giờ mình chỉ cần phải viết thêm khối else vào là xong, một cấu trúc if else được tạo ra. Việc thực thi ở phần if sẽ tương tự như câu lệnh if, nhưng nếu điều kiện ở if sai thì thay vì bỏ qua nó, trình biên dịch sẽ thực thi đoạn code bên trong else rồi mới tiếp tục đến với các dòng code sau dấu }.
* **c. Câu lệnh if – else if Ladder (if – else if Ladder Statement)**
* Dĩ nhiên không phải lúc nào các vấn đề trong cuộc sống cũng chỉ đơn giản sẽ là “nếu không cái này thì cái khác”, mà đa phần mình sẽ phải gặp các trường hợp “nếu không cái này thì cái khác và nếu không cái khác thì còn nhiều cái khác nữa” (khó ở thiệt sự ><) Trong lập trình cũng vậy, một bài toán phức tạp sẽ đặt ra rất rất nhiều các trường hợp và điều kiện khác nhau để có thể ra được kết quả cuối cùng, vì vậy mà ta cần phải làm quen với một câu lệnh if phức tạp hơn đó là **if – else if Ladder** (mình tạm dịch là “*cầu thang if else if”*) để giải quyết vấn đề nói trên.
* Một cấu trúc **if else if Ladder** sẽ được biểu diễn như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | if (điều kiện 1)  {     // Các lệnh ở khối if 1  }  else if (điều kiện 2)  {     // Các lệnh ở khối if 2  }  else if (điều kiện 3)  {     // Các lệnh ở khối if 3  }  ... // Có thể có thêm nhiều else if  else  {     // Các lệnh ở khối else  } |

* Trình biên dịch sẽ lần lượt check các điều kiện có trong các khối if ở trên, theo thứ tự từ trên xuống, nếu có một điều kiện nào thỏa mãn, chương trình sẽ thực thi code trong khối if đó và thoát ra khỏi cả khối if bự đó luôn, tức sẽ không xét các điều kiện if khác nếu vẫn còn.
* A screenshot of a computer

  Description automatically generated with low confidence

**d. if lồng if (Nested if Statement)**

* Các bạn cứ tưởng việc viết code bên trong khối if cũng giống như việc các bạn đang viết ở một cái main() khác vậy, vì thế mà việc có thể sử dụng thêm các **“if con”** bên trong đó là điều hiển nhiên rồi đúng không. Cấu trúc của một **“if lồng if” (*Nested if*)** trong C++ như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | if(điều kiện 1)  {      if (điều kiện 2)      {          // Lệnh của khối if nằm bên trong (if con)      }  } |

* Việc viết thêm bao nhiêu **“if con”** bên trong cũng không thành vấn đề gì, miễn là các bạn vẫn luôn đảm bảo đúng cú pháp, có dấu {}đầy đủ là ok rồi nha :3 Cái này mình sẽ không đi sâu chi tiết vì thật ra nó cũng không quá phức tạp đâu mà đúng không nè:3

A picture containing shape

Description automatically generated

# 2. Vòng lặp (Loops)

Logo, company name

Description automatically generatedNguồn ảnh: [wikimediacommons](https://commons.wikimedia.org/wiki/File:Loop_logo.png)

**Vòng lặp (Loops)** là một trong những khái niệm cơ bản trong bất kì ngôn ngữ lập trình nào, nó là một cú pháp giúp ta lặp đi lặp lại một quá trình (dĩ nhiên ở đây sẽ là một đoạn code) với **một số lần lặp cố định** hoặc với một **điều kiện dừng** cho trước.

Ví dụ, giả sử bây giờ bạn muốn in ra màn hình tầm 100 dòng hehe boiz. Nếu không có vòng lặp, bạn sẽ phải ngồi viết 100 dòng cout ra đúng không, nhưng giờ với vòng lặp ta sẽ chỉ cần 3 dòng code là có thể thực hiện công việc tưởng chừng mệt nhọc này rồi.

Giờ mình thực tế hơn tí nha, mình sẽ cho bạn một file excel chứa hàng trăm hàng triệu thông tin người dùng, và mình cần bạn giúp mình in tên từng người một trong danh sách đó ra màn hình. Nếu thông thường, ta sẽ ngồi gõ từng cout cho từng cái biến chứa tên người dùng một, việc làm đó có thể sẽ là một thảm họa với hầu hết tất cả mọi người >< NHƯNG, bây giờ với việc đã học về vòng lặp, ta sẽ chỉ cần ghi đúng một dòng cout bên trong câu lệnh vòng lặp với một số lần lặp ở đây sẽ là số thứ tự của người cuối cùng trong file excel là ngon lành cành đào liền luôn :3

## a. Vòng lặp for (for Loop)

**Vòng lặp for (for Loop)** là một vòng lặp cho phép lặp một quá trình với **một số lần lặp nhất định**. Cấu trúc của vòng lặp for trong C++ như sau:

|  |  |
| --- | --- |
| 1  2  3  4 | **for**(khởi tạo biến lặp;điều kiện dừng của biến lặp;cập nhật biến lặp)  {       // Phần code bên trong  } |

**Trong đó:**

* **Khởi tạo biến lặp**: là nơi để ta khai báo một biến lặpvới một giá trị bất kì, nhiệm vụ của nó sẽ như là một máy đếm để đếm số lần đã lặp của vòng lặp, giá trị của nó sẽ thay đổi khi một lần lặp được hoàn thành.
* **Điều kiện dừng của biến lặp**: là nơi để ta đặt một điều kiện dừng vòng lặp này lại thông qua việc sử dụng toán tử quan hệ cho biến lặp, nếu điều kiện không còn đúng nữa thì vòng lặp sẽ dừng lại.
* **Cập nhật biến lặp**: là nơi để ta cập nhật giá trị cho biến lặp sau mỗi lần hoàn tất một lần lặp bằng các toán tử gán.

Wow, cả một mớ kiến thức mới luôn, mình đảm bảo đọc suông như vậy chắc chắn sẽ rất khó hiểu nên mình sẽ đi ngay tiếp đến ví dụ, cụ thể sẽ là ví dụ lặp việc in dòng chữ hehe boiz ra màn hình nha các bạn:

|  |  |
| --- | --- |
| 1  2  3  4  5 | // Dòng lệnh sau sẽ in một 100 dòng chữ hehe boiz ra màn hình  **for**(**int** i = 1; i <= 100; ++i)  {      cout << "hehe boiz" << endl ;  } |

Mình lắp đúng cấu trúc ở trên nha, ở phần **khai báo biến lặp**, mình đã khai báo một biến kiểu int là i và gán cho nó giá trị bằng 1. Ở phần **đặt điều kiện** **cho biến lặp**, theo yêu cầu đề bài, mình cần in ra 100 dòng chữ nên điều kiện dừng cho biến lặp này đơn giản sẽ là <= 100. Cuối cùng, để có thể in ra 100 lần như vậy thì ta sẽ cần đặt phép toán ++i (toán tử tăng) cho phần **cập nhật biến lặp** – tức sau mỗi lần lặp i sẽ được tăng lên 1 đơn vị.

Diagram

Description automatically generatedNguồn ảnh: [programiz](https://www.programiz.com/cpp-programming/for-loop)

## b. Vòng lặp while (while Loop)

**Vòng lặp while (while Loop)**là một vòng lặp cho phép lặp một quá trình dựa vào **điều kiện lặp** cho trước. Với vòng lặp for, ta dùng để lặp một quá trình với **số lần lặp đã biết trước**, còn với vòng while này, ta sẽ dùng để lặp một quá trình mà ở đó ta **không biết trước được số lần lặp cụ thể**của đoạn code phía trong nó. Cấu trúc của một vòng lặp while trong C++ như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5 | // Đây là cấu trúc của một vòng lặp while  while(điều kiện lặp)  {      // Lệnh bên trong khối while  } |

Vòng while này nhìn có vẻ đỡ rắc rối hơn vòng for ban nãy ha các bạn ^^ Ở đây ta sẽ chỉ cần định nghĩa một điều kiện bất kì, có thể là dành cho biến lặp ta tạo sẵn bên ngoài hay bất kì một mô tả nào mà các bạn muốn, vòng while sẽ **không ràng buộc** phải là điều kiện của biến lặp như ở vòng for. Việc thực thi vòng while cũng sẽ khá dễ hiểu, đầu tiên trình biên dịch sẽ check điều kiện lặp ở vòng while trước, nếu đúng thì sẽ thực thi khối code bên trong và sau khi hoàn thành, nó lại tiếp tục check điều kiện cho đến khi điều kiện không còn đúng nữa thì dừng.

Để ví dụ cho vòng while này mình sẽ viết lại bài in ra 100 dòng “hehe boiz” như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | // Khởi tạo một biến lặp  int i = 1;    while(i <= 100)  {       cout << "hehe boiz" << endl ;       i += 1;  } |

Đầu tiên, mình khởi tạo một biến lặp bên ngoài để làm ‘máy đếm‘ cho điều kiện dừng (mọi người hay đặt tên biến lặp là i – iterable). Sau đó, mình sẽ đặt duy nhất một điều kiện dành cho biến lặp là i <= 100, tức là khi i vẫn còn <= 100 thì ta sẽ **lặp lại** việc in dòng hehe boiz. Ngoài ra còn có dòng i += 1 trong khối while này là để mình có thể cập nhật biến i, nếu không có dòng này thì vòng lặp này sẽ bị **lặp vô hạn** – tức sẽ gây lỗi cho chương trình của chúng ta ><.

Diagram

Description automatically generatedNguồn ảnh: [programiz](https://www.programiz.com/cpp-programming/do-while-loop)

## c. Vòng lặp do…while (do…while Loop)

**Vòng lặp do…while** **(do…while Loop)** là một vòng lặp với bản chất khá giống vòng lặp while, điểm khác ở đây là nó sẽ **thực hiện các dòng code bên trong khối do while trước** rồi sau đó mới xét đến điều kiện lặp, tức là ta sẽ luôn đảm bảo được rằng đoạn code bên trong sẽ được thực hiện ít nhất một lần! Cấu trúc của một vòng do while như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5 | // Đây là cấu trúc của một vòng lặp do while trong C++  do  {      // Code bên trong khối do while  } while (điều kiện lặp); |

Như mình đã nói ở trên, trình biên dịch sẽ thực hiện các đoạn code bên trong khối do while trước rồi sau đó mới check điều kiện lặp, nếu đúng thì lặp tiếp, không thì sẽ tự động thoát ra. Các bạn nhớ để ý là có dấu ; ngay sau điều kiện lặp nha, mình thấy nhiều bạn thường hay quên vụ này. Đây chỉ như là một cách viết ngược của while nên mình sẽ không làm thêm ví dụ ở đây nha các bạn ^^

Diagram

Description automatically generatedNguồn ảnh: [programiz](https://www.programiz.com/cpp-programming/do-while-loop)

À, còn một vấn đề mà mình nghĩ nhiều bạn sẽ thắc mắc, đó là nên sử dụng vòng lặp nào trong lúc lập trình ? Cái này thì tùy vào trường hợp cụ thể mà mình sẽ sử dụng nó sao cho phù hợp nha các bạn, mình sẽ **không ưu tiên dùng cái nào hết!** Sẽ có trường hợp dùng for tối ưu hơn là dùng while hay do while và ngược lại. Mình biết phần này mới đầu sẽ rất khó và các bạn cần phải cố gắng luyện tập thật nhiều các dạng bài khác nhau để lĩnh hội cho mình được cách sử dụng vòng lặp hiệu quả nhé.

# 3. Từ khóa break và continue (break & continue Statements)

### break

**Từ khóa break (break statement)** được dùng để **thoát** khỏi vòng lặp ngay cả khi vòng lặp vẫn còn có thể lặp tiếp được. Khi trình biên dịch bắt gặp từ khóa break này, mọi đoạn code vẫn còn bên trong khối sẽ **bị bỏ qua** và **“đá”** trình biên dịch ra ngoài vòng lặp ngay và luôn. Từ khóa này sẽ đặc biệt phù hợp giúp ta**tránh** việc bị lặp vô hạn hay phải làm việc với một vòng lặp vô hạn thật sự.

Ban nãy thì mình bảo các bạn là phải tránh việc tạo ra một vòng lặp vô hạn, giờ thì mình lại nói về việc làm việc với một vòng lặp vô hạn :)) (hơi bị mâu thuẫn ha). Nhưng mà thật là như vậy, sẽ có những vòng lặp mà ta cũng sẽ không biết điều kiện để dừng là gì – hay nói đúng hơn là việc dừng đó sẽ **không do ta định đoạt** mà **phụ thuộc vào các yếu tố khác** (có thể hiểu là người dùng chương trình của chúng ta). Đây là cấu trúc của một vòng lặp vô hạn và cách sử dụng break:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | int i = 1;  int sum = 0;    // Đây là cách để tạo ra một vòng lặp vô hạn  while(true)  {      sum += i ;      i += 1;        // Vòng lặp sẽ thoát khi tổng lớn hơn hoặc bằng 100      if (sum >= 100)      {          break;      }    }    cout << sum << endl ; // Kết quả là 105 |

Nói qua một chút về cách triển khai một vòng lặp vô hạn, ở đây mình dùng vòng lặp while với điều kiện là true, rất lạ phải không. Mình sẽ không dùng bất cứ toán tử quan hệ nào để check tính đúng sai của điều kiện mà mình sẽ **đặt luôn giá trị là**true để điều kiện lặp này **luôn đúng** – đồng nghĩa vòng lặp này giờ đây sẽ được lặp vô hạn.

Quay lại với break, ở đoạn code trên mình đã dùng nó để làm một điều kiện dừng cho vòng lặp vô hạn này, khi điều kiện bên trong if đúng – tức khi sum >= 100, trình biên dịch sẽ gặp từ khóa break và ngay lập tức thoát khỏi vòng lặp trên, sau đó gặp dòng cout và ra cho ta giá trị của sum – tức 105.

Graphical user interface, application, Word

Description automatically generatedNguồn ảnh: [programiz](https://www.programiz.com/c-programming/c-break-continue-statement)

### continue

**Từ khóa continue (continue statement)** cũng được dùng khi làm việc với vòng lặp, mục đích của nó dùng để **ngay lập tức bỏ qua** việc thực thi code của lần lặp hiện tại, **nhảy lên check điều kiện lặp** và thực hiện lần lặp tiếp theo (nếu có). Đây là ví dụ minh họa cách sử dụng từ khóa continue:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | // Đây là chương trình tính tổng các số chẵn trong khoảng từ 1 đến 100  int sum = 0;    for(int i = 1; i <= 100; ++i)  {      if (i%2 != 0) // Nếu biến lặp là một số lẻ thì chuyển qua lần lặp tiếp theo      {         continue ;      }      else // Nếu biến lặp là một số chẵn thì cộng vào biến sum      {         sum += i ;      }  } |

Đoạn code trên là chương trình tính tổng các số chẵn trong khoảng từ 1 đến 100 nên mình sẽ có 100 lần lặp. Để có thể check số hiện tại là một số lẻ thì mình chỉ cần chia lấy dư số đó cho 2, nếu nó **không dư 0** thì tức nó là một số lẻ (kiến thức toán cơ bản nè). Mỗi lần lặp, ta sẽ check xem nếu giá trị của biến lặp là một số lẻ, ta sẽ dùng từ khóa continue để ngay lập tức đẩy đến lần lặp tiếp theo, còn nếu không phải thì cộng dồn vào biến sum – vì không là số lẻ thì chắc chắn là số chẵn nên không nhất thiết phải viết thêm điều kiện đâu nha.

Graphical user interface, application

Description automatically generatedNguồn ảnh: [programiz](https://www.programiz.com/c-programming/c-break-continue-statement)

# 4. Câu lệnh switch…case (switch Statements)

Đây cũng là một cấu trúc điều khiển trong C++ và mục đích hoạt động của nó hoàn toàn tương tự như cấu trúc if – else if Ladder – nó cho phép ta check dữ liệu của một biến và ứng với từng giá trị cụ thể (**case**), ta sẽ có các kết quả trả về khác nhau. Một cấu trúc **switch…case** trong C++ sẽ được viết như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | switch (biến)  {     case <điều kiện 1>:        // Code ở case 1        break;     case <điều kiện 2>:        // Code ở case 2:        break;     ...     case <điều kiện n>:        // Code ở case n        break;       default:        // Code ở phần không thuộc bất kì case nào (mặc định)  } |

**Cấu trúc switch…case** ***(switch statement)*** sẽ check lần lượt các case theo thứ tự **từ trên xuống (giống if – else if Ladder)**, nếu giá trị của biến bằng giá trị của một trong các case trên thì sẽ thực thi đoạn code sau dấu :, nếu không dính bất cứ case nào thì nó sẽ thực hiện code ở default (nếu có). Điểm đặc biệt các bạn cần lưu ý ở switch…case là ta sẽ **có từ khóa** break**sau mỗi cuối phần code bên trong một** case (trừ default vì nó đã mặc định nằm sẵn bên dưới cùng của một switch…case), không giống như if – else if Ladder khi chỉ cần đúng một điều kiện là sẽ bỏ qua việc xét các điều kiện còn lại, switch…case sẽ thực thi hết đoạn code của các case đúng mà không tự động thoát ra, vì vậy mà ta luôn cần phải có từ khóa break để thoát khỏi việc check các case còn lại bên trong switch (khá giống vòng lặp).

Việc sử dụng switch…case hay if – else if Ladder là tùy thuộc vào các bạn, vì mục đích sử dụng của nó là như nhau. Nhưng theo một số lời khuyên thì việc sử dụng switch…case sẽ làm cho code của bạn **dễ đọc hơn** nên các bạn có thể cân nhắc nhé ^^

Diagram

Description automatically generated

Mảng tĩnh

**Mảng một chiều**

Mảng là một kiểu dữ liệu chứa các phần tử cùng loại được chứa trong một khối bộ nhớ liên tiếp. Trong một mảng A có kích thước N , mỗi vị trí bộ nhớ được đánh một chỉ số duy nhất i với điều kiện 0<= i <N, tức là chỉ số của mảng luôn luôn được bắt đầu từ 0 và phải nhỏ hơn kích thước của mảng. Để truy cập đến mỗi phần tử của mảng ta chỉ cần biết chỉ số của chúng, khi ta viết A[i] có nghĩa là ta đang truy cập tới phần tử thứ i của mảng A.

**Ví dụ:**

Ta có mảng A với kích thước là 8 phần tử, phần tử đầu tiên của mảng được bắt đầu từ vị trí ô nhớ có địa chỉ là 1200

|  |  |  |
| --- | --- | --- |
| **Địa chỉ** | **Giá trị** | **Phần tử mảng** |
| 1200 | f | A[0] |
| 1201 | h | A[1] |
| 1202 | h | A[2] |
| 1203 | a | A[3] |
| 1204 | y | A[4] |
| 1205 | t | A[5] |
| 1206 | I | A[6] |
| 1207 | a | A[7] |

**Chú ý:**

* Kích thước của một mảng phải được khai báo trước khi ta chèn dữ liệu vào mảng đó.
* Các phần tử trong mảng phải có cùng kiểu dữ liệu.
* Trong C++ kích thước của một mảng tĩnh sau khi khai báo là cố định và không thể thay đổi được, vì vậy ta không thể khai báo kích thước của một mảng tĩnh là biến được.

**Ví dụ:**

Trong trường hợp sau trình biên dịch sẽ báo lỗi.

int N = 100;

int A[N];

Nhưng đoạn mã sau hoàn toàn có thể chạy được.

#define N 100

int A[N];

Sự khác nhau ở đây là do tính chất của C++. Với trường hợp thứ nhất, giá trị của N là không xác định từ thời điểm biên dịch tới thời điểm chạy runtime. Do vậy tại thời điểm biên dịch trình biên dịch sẽ báo lỗi do kích thước của mảng không được chỉ rõ.

Ở trường hợp thứ 2 do chúng ta sử dụng một kỹ thuật đặc biệt trong lập trình đó là tiền xử lý. Khi này tại thời điểm biên dịch chương trình, giá trị của N đã được xác định, đồng nghĩa với việc trình biên dịch đã biết được kích thước của mảng nên lỗi biên dịch sẽ không xảy ra ở đây.

Một số điểm cần chú ý khi làm việc với mảng:

|  |
| --- |
| Coi kích thước của mảng là N.  Thời gian truy cập tới một phần tử của mảng:   * O(1) - Điều này là có thể vì tất cả các phần tử của mảng được chứa trong các vị trí ô nhớ liên tiếp.   Thời gian tìm kiếm:   * O(N) trong trường hợp tìm kiếm tuần tự (Sequential Search). * O(log N) cho trường hợp tìm kiếm nhị phân (Binary Search) và nếu mảng đã được sắp xếp từ trước.   Thời gian chèn dữ liệu vào mảng:   * O(N) - Trường hợp xấu nhất xảy ra khi ta muốn chèn dữ liệu vào vị trí đầu tiên của mảng và phải thực hiện dịch toàn bộ phần tử tiếp theo của mảng, trường hợp tốt nhất xảy ra khi ta muốn chèn một phần tử vào vị trí cuối của phần dữ liệu trong mảng.   Thời gian cần thiết để xóa một phần tử trong mảng:   * O(N) Xảy ra khi ta muốn xóa phần tử đầu tiên của mảng và đòi hỏi ta phải thực hiện dịch tất các các phần tử còn lại của mảng về vị trí có chỉ số nhỏ hơn chỉ số hiện tại của chúng 1 đơn vị, đây cũng là trường hợp xấu nhất. |

**Mảng hai chiều**

Mảng hai chiều có đầy đủ các tính chất của bảng một chiều như:

* Dữ liệu được điền trong một khối các ô nhớ liên tiếp.
* Các phần tử trong cùng một mảng phải có cùng kiểu dữ liệu.
* Kích thước của mảng hai chiều phải được khai báo trước khi ta chèn dữ liệu vào bảng đó (chính xác hơn, kích thước của mảng phải được xác định trong thời gian biên dịch chương trình).

**THêm xóa phần tử**

## Xây dựng các hàm nhập xuất mảng

Đầu tiên, bạn hãy xử lý xong phần nhập xuất mảng nhé. Chúng ta bắt buộc phải sử dụng nhập/ xuất mảng để sử dụng trong quá trình thêm xóa phần tử mảng.

Code hàm nhập và hàm xuất mảng:

Text

Description automatically generated

## Thêm phần tử vào mảng 1 chiều

Bạn có thể hình dung cách thêm 1 phần tử vào mảng qua hình ảnh dưới đây nhé.

Diagram

Description automatically generated

Như vậy, trước khi thực hiện thêm, bạn phải thực hiện lùi tất cả các số phía sau nơi được chèn(bao gồm cả vị trí chèn) về phía sau 1 ô nhớ. Như ảnh trên thì chỉ có số 9 bị ảnh hưởng, bị chuyển từ chỉ số 2 sang 3.

Sau khi dịch chuyển theo mô tả trên, chúng ta chỉ việc gán giá trị cần chèn cho phần tử tại nơi muốn chèn. Và cuối cùng, không quên tăng số lượng phần tử của mảng lên nhé.

Quy trình thêm phần tử vào mảng:

1. Kiểm tra mảng có thể thêm được phần tử nữa không? Nếu không, thoát hàm
2. Kiểm tra giá trị pos hợp lệ không. Ở đây nếu không hợp lệ mình cho về chỉ số đầu/cuối.
3. Thực hiện dịch chuyển mảng(phần phía sau nơi chèn + vị trí chèn)
4. Chèn vào vị trí cần chèn
5. Tăng số lượng phần tử

Text

Description automatically generated

## Xóa phần tử trong mảng 1 chiều

Bạn có thể hình dung cách xóa 1 phần tử trong mảng qua hình ảnh dưới đây nhé.

Table

Description automatically generated

Dễ nhận thấy, số lượng phần tử sẽ giảm đi 1. Tuy nhiên, hãy để ý các số từ 67 trở về sau đều bị giảm chỉ số đi 1 đơn vị.

Quy trình xóa phần tử trong mảng:

1. Kiểm tra có thể xóa hay không? Nếu không => thoát hàm
2. Kiểm tra giá trị pos hợp lệ không. Ở đây nếu không hợp lệ mình cho về chỉ số đầu/cuối.
3. Dịch chuyển mảng lùi 1 chỉ số - phần phía sau nơi xóa
4. Giảm số lượng phần tử

Text

Description automatically generated

**Tại sao phải tách file?**

Khi lưu toàn bộ code vào một file main sẽ gây ra rất nhiều vấn đề bất lợi, có thể kể đến như: Chương trình trở nên quá dài; khó quản lý, sử dụng. Giả sử ta khi muốn tìm một hàm nào đấy để chỉnh sửa thì sẽ làm tiêu tốn thời gian vì có quá nhiều hàm và không biết nó ở đâu.

Việc tách code C++ cũng như tách code ở các ngôn ngữ khác sẽ giúp:

* Dễ quản lí, bảo trì source code.
* Giúp code dễ đọc, dễ hiểu và dễ sử dụng.
* Tái sử dụng các hàm đã viết.

## Mã code các file

**Một chút về file Header (.h)**: Bạn có thể tưởng tượng đây là một file thư viện mà mình tự viết ra. Nó chứa các khai báo hàm, định nghĩa marco và có thể chia sẽ sang các file khác. (Trong các trình biên dịch sẽ có các thư viện - file Header được nhà sản xuất viết sẵn như string.h, time.h, math.h, …)

**Note: Mỗi file .h đều phải có dòng code #pragma once ở đầu**.

### **a) File: Header.h**

Đối với mình, đây sẽ là file include các thư viện mà tất cả các file khác đều phải có. Phổ biến nhất là . Nhằm mục đích tái sử dụng, không cần include một thư viện nhiều lần. Ngoài ra có thể khai báo các các class, struct cần thiết vào. Thậm chí là cả using namespace std

### **b) File: Math\_Fn.h và Math\_Fn.cpp**

### **c) File: String\_Fn.h và String\_Fn.cpp**

### **d) File: main.cpp**

Include tất cả các file .h (header) đã viết vào, rồi viết hàm main.