ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN NGUYÊN LÝ VÀ PHƯƠNG PHÁP LẬP TRÌNH

Đề tài : Biến trong ngôn ngữ lập trình

Nguyễn Tường Vy – 19522548

Lớp : CS111.M22.KHCL

GVHD:Trịnh Quốc Sơn

MỤC LỤC

LỜI CẢM ƠN

Đầu tiên, em xin gửi lời cảm ơn chân thành đến quý thầy cô giảng viên Trường Đại học Công nghệ thông tin – Đại học Quốc gia TP. Hồ Chí Minh đã giúp cho em có những kiến thức cơ bản làm nền tảng để thực hiện đề tài này.

Đặc biệt, em xin gửi lời cảm ơn và lòng biết ơn sâu sắc nhất tới thầy giáo Trịnh Quốc Sơn, người đã hướng dẫn cho em suốt thời gian làm đề tài. Thầy đã trực tiếp hướng dẫn tận tình, sửa chữa và đóng góp nhiều ý kiến quý báu giúp em hoàn thành tốt báo cáo môn học của nhóm. Một lần nữa em chân thành cảm ơn thầy và chúc thầy dồi dào sức khỏe.

Trong thời gian một học kỳ thực hiện đề tài, em đã vận dụng những kiến thức nền tảng đã tích lũy đồng thời kết hợp với việc học hỏi và nghiên cứu những kiến thức mới từ thầy cô, bạn bè cũng như nhiều nguồn tài liệu tham tham khảo. Từ đó, nhóm chúng em vận dụng tối đa những gì đã thu nhập được để hoàn thành một bài tiểu luận tốt nhất. Tuy nhiên, vì kiến thức chuyên môn còn hạn chế và bản thân còn nhiều thiếu xót kinh nghiệm thực tiễn nên nội dung của báo cáo không tránh khỏi những thiếu xót, em rất mong nhận được sự góp ý, chỉ bảo thêm của quý thầy cô nhằm hoàn thiện những kiến thức của mình để em có thể dùng làm hành trang thực hiện tiếp các đề tài khác trong tương lai cũng như là trong việc học tập và làm việc sau này.

Một lần nữa xin gửi đến thầy cô, bạn bè lời cảm ơn chân thành và tốt đẹp nhất!

Thành phố Hồ Chí Minh, tháng 6 năm 2022
Sinh viên thực hiện

1.Nội dung đồ án :

1.1 Giới thiệu :

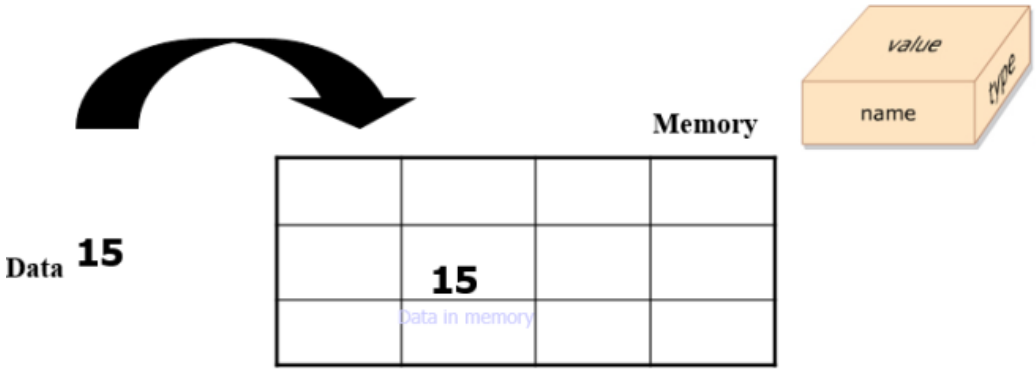
- Biến (Variables) là phần không thể thiếu trong bất kì ngôn ngữ lập trình nào, nếu không có biến thì ta rất khó hoàn thành được bất kì chương trình nào cả, hầu như là không thể .

Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00	sum	int (4 bytes)	000000FF(255_{10})
90000001	00			
90000002	00			
90000003	FF			
90000004	FF	age	short (2 bytes)	FFFF(-1_{10})
90000005	FF			
90000006	1F	averge	double (8 bytes)	1FFFFFFFFFFFFFFF ($4.45015E-308_{10}$)
90000007	FF			
90000008	FF			
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF			
9000000D	FF			
9000000E	90	ptrSum	int* (4 bytes)	90000000
9000000F	00			
90000010	00			
90000011	00			

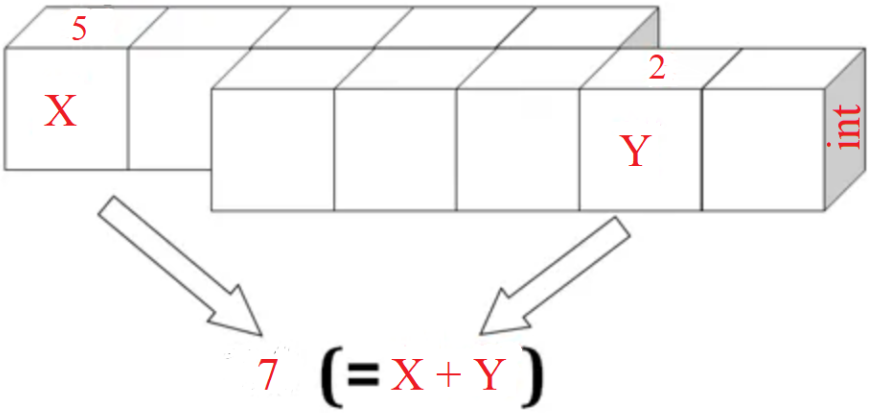
Note: All numbers in hexadecimal

1.2 Khái niệm biến là gì?

- Trong lập trình, một biến là một giá trị có thể thay đổi, tùy thuộc vào điều kiện hoặc thông tin được truyền đến chương trình. Thông thường, một chương trình bao gồm các lệnh cho máy tính biết phải làm gì và dữ liệu mà chương trình sử dụng khi nó đang chạy. Dữ liệu bao gồm các hằng số hoặc giá trị cố định không bao giờ thay đổi và các giá trị biến (thường được khởi tạo thành "0" hoặc một số giá trị mặc định vì các giá trị thực tế sẽ do người dùng chương trình cung cấp).



Ví dụ trong ngôn ngữ C++ để in ra màn hình kết quả của 5+2 ,ta sẽ gán X với kiểu dữ liệu ***int*** *tương ứng với 5 , tương tự với Y , sau đó dùng lệnh* ***cout<<X+Y;***



1.3 Tên biến trong ngôn ngữ lập trình :

- Tên của biến là một chuỗi kí được dùng để định danh một vùng nhớ chứa một giá trị mà người dùng nhập vào . Vậy vì sao cần phải đặt tên biến :
  - *Đặt tên đúng giúp mã nguồn dễ đọc hơn, dễ sử lý hơn.*
  - *Nó cũng là cái nguyên tắc của lập trình viên, bạn tự nhiên làm trái là không được rồi.*
  - *Giúp lập trình trở nên dễ dàng hơn, bạn dễ dàng nhớ tác dụng của biến, hàm, class*
  - *Code của bạn trở nên khoa học hơn, theo đúng một chuẩn mực nào đó.*
- Độ dài tên biến phụ thuộc vào ngôn ngữ lập trình nào mà người dùng sử dụng ,kiểu dữ liệu của biến .

Tên ngôn ngữ lập trình	Độ dài tối đa
FORTRAN I	*6*
FORTRAN 90	*31*
ANSI C	
Python	*79*
Ada	*No limit*
Java	
C/C++	*No limit*

- Về quy tắc đặt tên đa phần các ngôn ngữ lập trình có chung quy tắc:

➢ *Tên không được bắt đầu bằng chữ số. Nó phải được bắt đầu bằng dấu gạch dưới _ hoặc một kí tự chữ.*

➢ *Tên không được đặt trùng với từ khóa của ngôn ngữ. (Mỗi ngôn ngữ từ khóa lại khác nhau .*

Ví dụ : Ngôn ngữ C/C++:

```
int main()
{
    int sotuoi;
    cin>> sotuoi;
    cout<<"so tuoi cua ban la : "<<
}
```

```
int main(void){
  int num = 7;
  char str[] = " con cá";
  printf("%d%s", num , str );
}
```

Ngôn ngữ Fortran :

```
character (len = 15) :: name
integer   :: id
real      :: weight
name = 'Ardupilot'
```

- Case Sensitive (Phân biệt chữ viết hoa và viết thường) :
  - *Những kí tự hoa sẽ khác những kí tự thường*

  VD : Rose , RoSe , ROse , cả ba tên này đều khác nhau
  - *Tuy khó khăn trong việc phân biệt khi đọc tên biến , nhưng case sensitive giúp cho không gian tên lớn hơn, có thể sử dụng để biểu thị các lớp của biến .*
  - *Không phải ngôn ngữ lập trình nào cũng có case sensitive , một số ngôn ngữ sử dụng case sensitive : C, C++, Java, and Modula-2.*

-Bên cạnh những biến được định danh theo người lập trình , theo ngôn ngữ lập trình thì cũng tồn tại những biến không có tên .Một số ngôn ngữ lập trình (bao gồm một số ngôn ngữ hợp ngữ) có khái niệm về các ***đối số ngầm định (Implicit Arguments)*** . Đối số ngầm định cho phép bạn truy cập và hoặc sửa đổi một giá trị mà không cần đặt tên rõ ràng cho đối số.Đối số vẫn có tên, ngay cả khi bạn không phải cung cấp nó.

Ví dụ :Trong Perl, bạn có thể in toàn bộ một tệp, thay đổi tất cả các phiên bản của "Betty" thành "Wilma" bằng một chương trình như sau:

```
while (<>) {
    s/Betty/Wilma/g;
    print;
}
```

Chương trình này đề cập đến hai biến riêng biệt một cách ngầm định: `<>` tương đương với `<STDIN>`

Trong C ++, bạn có thể tham chiếu đến các biến member của một lớp từ

bên trong các hàm ,member, mà không có bất kỳ loại tiền tố nào

```
class clown {
  private:
    int member;

  public:
    method() { member++; }
};
```

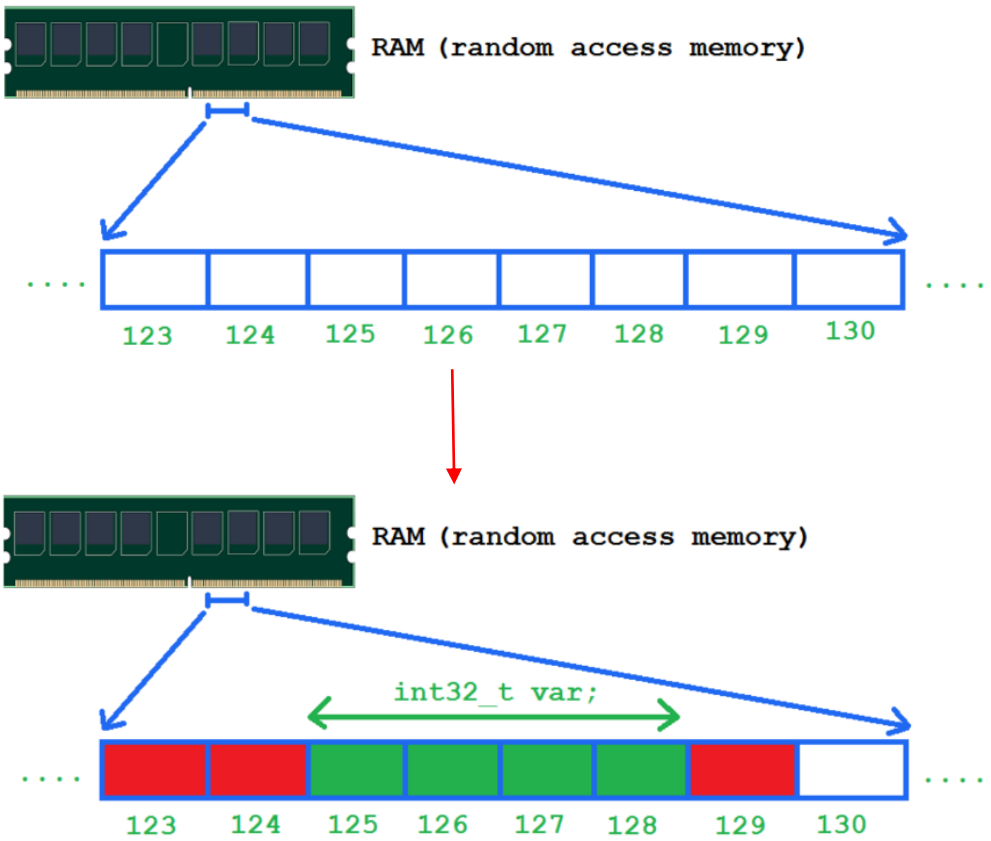
Tuy nhiên,có thể có nhiều trường hợp của một lớp. Làm thế nào để mã đã biên dịch biết phiên bản nào cần cập nhật?Vậy nên để giải quyết vấn đề này C ++ định nghĩa một biến có tên `this` - truy cập đến các thuộc tính, phương thức của đối tượng hiện tại của một lớp khi gọi một hàm member trên một đối tượng.Nhưng thay vì `this->member++` thì có thể viết `member++`. Hầu hết thời gian, điều này vẫn bị ẩn.

-Ta có thể nhận thấy tất cả các biến này đều có tên,cú pháp.Một số biến cho phép chúng ta bỏ qua tên, nhưng điều đó không có nghĩa là biến đó không có tên.

1.4 Địa chỉ của biến :

- RAM hay các thiết bị cung cấp bộ nhớ tạm thời khác đều được tạo nên bởi các ô nhớ liên tiếp nhau, mỗi ô nhớ đều có 1 số thứ tự đại diện cho vị trí của ô nhớ đó trong thiết bị lưu trữ. Chúng ta có thể gọi con số đó địa chỉ của ô nhớ.
- Những địa chỉ của ô nhớ chỉ là những con số ảo được tạo ra do hệ điều hành, còn về bản chất bên trong việc quản lý bộ nhớ của máy tính thì máy tính của chúng ta có những thiết bị riêng để làm điều đó.
- Các bạn cứ tưởng tượng 1 ô nhớ trong thiết bị lưu trữ là một cái nhà trên con đường, để xác định được vị trí của 1 cái nhà, chúng ta cần biết địa chỉ của nhà cần tìm.

*Ví dụ : Giả sử biến **var** được khai báo bằng kiểu dữ liệu **int32_t**, và hệ điều hành tìm được vùng nhớ trống đủ 4 bytes để cung cấp cho biến **var** tại vị trí 125 đến 128, biến var sau khi được cấp phát vùng nhớ sẽ có địa chỉ 125 (là địa chỉ của ô nhớ đầu tiên mà biến nắm giữ).*



-Một tên biến có thể có các địa chỉ khác nhau ở những nơi khác nhau và tại thời gian khác nhau trong quá trình thực hiện.

Ví dụ : Khi chạy đoạn chương trình sau :

```
 9  #include <iostream>
10
11  using namespace std;
12
13  int main()
14  {
15      int32_t var;
16      cout << "Address of var: " << &var << endl;
17  }
```

```
Address of var: 0x7fffc3c018d4
```

```
Address of var: 0x7ffc3e2bec04
```

Chúng ta thấy qua 2 lần chạy chương trình thì địa chỉ của biến này có 2 vị trí khác nhau. Đồng nghĩa với việc chọn vị trí vùng nhớ để cấp phát cho biến hoàn toàn được thực thi tự động bởi hệ điều hành.

1.5 Alias :

- Nếu hai hay nhiều tên biến được sử dụng để truy cập cùng một bộ nhớ vị trí, chúng được gọi là Alias .
- Alias được tạo thông qua con trỏ, biến tham chiếu, C và C ++ các đoàn thể.

Có thể hiểu một biến tham chiếu là một ***Alias***, đó là tên khác cho một biến đã đang tồn tại. Khi một tham chiếu được khởi tạo với một biến, thì: hoặc tên biến hoặc tên tham chiếu có thể được sử dụng để tham chiếu tới biến đó.

- Việc có nhiều alias trong chương trình thường sẽ gây bối rối cho người dùng vì phải nhớ tên các alias đó .

Ví dụ : Ở đoạn chương trình sau :

```
int i;
int&r = i;
```

*Dấu **&** trong khai báo này là **tham chiếu**. Vì thế, trong khai báo đầu tiên, r là tham chiếu integer được khởi tạo cho I, có thể gọi r là một alias .*

1.6 Kiểu dữ liệu của biến :

- Kiểu dữ liệu của biến dùng để xác định phạm vi giá trị của các biến và tập hợp

hoạt động được xác định cho các giá trị của loại đó .

- Dung lượng bộ nhớ được cấp cho biến bởi hệ điều hành phụ thuộc vào kiểu dữ liệu được lưu trữ trong biến.
- Để chỉ định bộ nhớ cho một đơn vị dữ liệu, chúng ta phải khai báo một biến với một kiểu dữ liệu cụ thể.

-Kiểu dữ liệu thường được dùng trong các công cụ lập trình có thể được phân chia thành :

- *Kiểu dữ liệu số – lưu trữ giá trị số.*
- *Kiểu dữ liệu ký tự – lưu trữ thông tin mô tả*

- Những kiểu dữ liệu này có thể có tên khác nhau trong các ngôn ngữ lập trình khác nhau.

Ví dụ : Một kiểu dữ liệu số được gọi trong C là ***int*** *trong khi đó tại Visual Basic được gọi là* ***integer****. Một kiểu dữ liệu ký tự được đặt tên là* ***char*** *trong C trong khi đó trong Visual Basic nó được đặt tên là* ***string****.Đặc biệt trong Python khi khai báo biến kiểu dữ liệu của nó sẽ* ***tự động được detect*** *.*

-Trong bất cứ trường hợp nào, các dữ liệu được lưu trữ luôn giống nhau. Điểm khác duy nhất là các biến được dùng trong một công cụ phải được khai báo theo tên của kiểu dữ liệu được hỗ trợ bởi chính công cụ đó.

-Thông tin một số kiểu dữ liệu:

Kiểu dữ liệu	**Ý nghĩa**	**Kích thước**	**Phạm vi**
short	*Số nguyên*	*2 bytes*	*-32,768 đến 32,767*
int		*4 bytes*	*-2,147,483,648 đến 2,147,483,647*
unsigned short	*Số nguyên dương*	*2 bytes*	*0 đến 65,535*
unsigned int		*4 bytes*	*0 đến 4,294,967,295*
float	*Số thực*	*4 bytes*	*3.4E +/- 38 (7 chữ số)*
double		*8 bytes*	*1.7E +/- 308 (15 chữ số)*
bool	*Kiểu luận lý*	*1 bytes*	*False or true*
char	*Số nguyên hoặc ký tự*	*1 bytes*	*-128 đến 127*
unsigned char		*1 bytes*	*0 đến 255*

C/C+

Kiểu dữ liệu	Ý nghĩa	Kích thước	Phạm vi
short	*Số nguyên*	*2 bytes*	*-32,768 đến 32,767*
int		*4 bytes*	*-2,147,483,648 đến 2,147,483,647*
byte		*1 bytes*	*-128 đến 127*
long		*8 bytes*	*-9,223,372,036,854,775,808 đến 9,223,372,036,854,775,807*
float	*Số thực*	*4 bytes*	*3.4E +/- 38 (7 chữ số)*
double		*8 bytes*	*1.7E +/- 308 (15 chữ số)*
boolean	*Kiểu luận lý*	*1 bit*	*False or true*
char	*Số nguyên hoặc ký tự*	*2 bytes*	*0 đến 65535*

Kiểu dữ liệu	*Ý nghĩa*
long	*Số nguyên*
int	
complex	*Số phức*
float	*Số thực*
tuple	*Kiểu Chuỗi*
list	
str	*Tập kí tự*

Python

1.7 Giá trị của biến :

- Ngoài là nội dung mà biến đó chứa thì giá trị biến còn dùng để xác định kiểu dữ liệu cho hợp lí .

Ví dụ : Vì tuổi thọ trung bình của con người là 80 tuổi , nên ngoài int ta có thể dùng kiểu dữ liệu short sẽ giúp giảm bớt dung lượng vùng nhớ .

```
int main()
{
    int sotuoi;
    cin>> sotuoi;
    cout<<"so tuoi cua ban la : "<<
}
```

1.7.1 L-value:

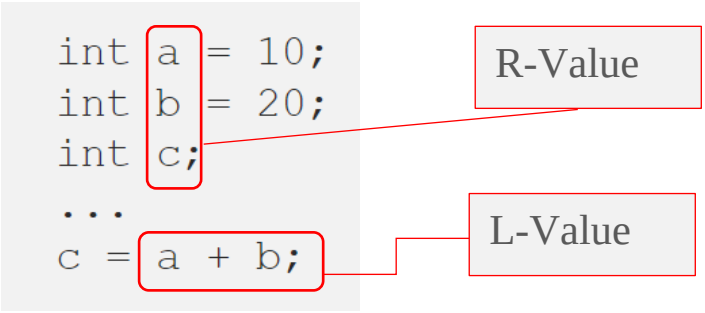
- *L-value là một Named Object nó có thể là Modifiable hoặc Non-modifiable.vbc*
- *Named Object là một vùng nhớ mà bạn đã đặt một symbolic name cho biến (nói cách khác: Named Object là một biến (variable).*
- *Literal không có name (cái mà chúng ta gọi là value-type), do đó nó không thể là một L-value.*

➢ Const variable là một non-modifiable L-value, nhưng bạn không thể đặt chúng ở bên trái của một lệnh gán. Dó đó chúng ta có luật:" Chỉ modifiable l-value có thể ở bên trái của một lệnh gán".

1.7.2 R-value:

➢ Một R-value là một un-named object; hoặc là một temporary variable.

Ví dụ :



Ví dụ :

```
int a = 0;
int b = 0;
...
a = b ++++; // trái sang phải, tăng sau khi gán
```

```
int a = 0;
int b = 0;
...
a = ++++ b; // phải sang trái, tăng trước khi gán
```

```
main.cpp:11:7: error: lvalue required as increment operand
    b++++;
```

Sau khi compile hai đoạn code đã cho, thì đoạn A bị lỗi ,đoạn B chạy được vì :

Thứ nhất toán tử tăng trước có thứ tự ưu tiên từ phải qua trái. Có nghĩa `++++b = ++(++b)`. *Do biểu thức* `++b` *trả lại*

một tham chiếu (tham chiếu) đến giá trị vừa được tăng (sau lệnh ++b) - giá trị này đang được lưu trong biến b, bởi vậy nó là l-value, nên được phép toán ++(l-value)đúng.
Toán tử tăng sau có ưu tiên thứ tự từ trái qua phải. Có nghĩa b++++ = (b++)++. Do biểu thức b++trả lại một bản sao của cái cũ giá trị của b trước khi tăng bởi câu lệnh b ++, bản sao giá trị này là một temporary value- chỉ tồn tại trong câu lệnh, như vậy nó (b++) là một r-value. Và như message thông báo ***"lvalue required as increment operand"*** *— toán hạng của phép toán increment phải là l-value, nên biểu thức (r-value)++ là sai.*

1.8 Liên kết (Binding) :

1.8.1 Thời gian liên kết(Binding time) :

- Thời gian ràng buộc là thời gian mà một liên kết diễn ra.Các thời gian liên kết có thể xảy ra:

- *Ngôn ngữ thiết kế thời gian: liên kết các ký hiệu toán tử với các phép toán.Ví dụ: Biểu tượng dấu hoa thị (*) được liên kết với phép toán nhân.*
- *Thời gian thực hiện ngôn ngữ: Ví dụ : Kiểu dữ liệu int trong C được liên kết đến một phạm vi chứa các giá trị nhất định.*
- *Thời gian biên dịch: :liên kết một biến với một kiểu dữ liệu cụ thể khi biên dịchthời gian*
- *Thời gian tải: liên kết một biến với một ô nhớ (ví dụ: biến tĩnh trong C)*
- *Runtime: liên kết một biến cục bộ không ổn định(nonstatic local variable) với một ô nhớ.*

1.8.2 Liên kết thuộc tính với các biến (Binding of Attributes to Variables) :

- Một liên kết là tĩnh (static) nếu nó xuất hiện lần đầu trước thời gian chạy và không thay đổi trong suốt quá trình thực thi chương trình.

- Một liên kết là động (dynamic) nếu nó xuất hiện lần đầu tiên trong quá trình thực thi hoặc có thể thay đổ trong quá trình thực hiện chương trình.

1.8.3 Các kiểu liên kết :

1.8.3.1 Liên kết tĩnh :

- Nếu là liên kết tĩnh (static) thì có thể được chỉ định bởi một khai báo tường minh hoặc ngầm định :
- Một khai báo tường minh là một câu lệnh chương trình được sử dụng để khai báo các loại biến.
- Một khai báo ngầm định là một cơ chế mặc định để chỉ định cácloại biến (sự xuất hiện đầu tiên của biến trong chương trình.)
- Cả khai báo tường minh và ngầm định đều tạo ra các ràng buộc tĩnh đối với các kiểu.
- FORTRAN, PL / I, BASIC và Perl cung cấp các khai báo ngầm định.

Ví dụ :

- *Trong Fortran, một số nhận dạng xuất hiện trong một chương trình không được khai báo rõ ràng sẽ khai báo ngầm theo như quy ước:I, J, K, L, M hoặc N hoặc các phiên bản viết thường của chúng được khai báo ngầm là kiểu* ***integer*** *hoặc kiểu* ***real****.Vì thế khi các vars vô tình không được khai báo sẽ được đưa ra các loại mặc định và các thuộc tính không mong muốn, có thể gây ra những lỗi khó chẩn đoán.*
- *Với ngôn ngữ lập trình Perl: Các tên bắt đầu bằng $ là một đại lượng vô hướng, nếu một tên bắt đầu bằng @ nó là một mảng, nếu nó bắt đầu bằng %, nó là một cấu trúc băm.Trong trường hợp này, tên @apple và %apple là khác nhau*

-Trong C và C ++, người ta phải phân biệt giữa khai báo và định nghĩa.

1.8.3.2 Liên kết động :

-Trong liên kết động , trình biên dịch không giải quyết ràng buộc tại thời gian biên dịch. Liên kết xảy ra trong thời gian chạy. Nó còn được gọi là ràng buộc muộn. Binding động xảy ra trong phương thức ghi đè.

- Ưu điểm:
  - *Linh hoạt*
  - *Đáp ứng nhu cầu về các loại "đa hình"*
  - *Phát triển các chức năng chung (ví dụ: sắp xếp)*
- Nhược điểm:
  - *Các ràng buộc kiểu động phải được thực hiện bằng cách sử dụng thông dịch viên không phải trình biên dịch.*
  - *Giải thích thuần túy thường mất ít nhất mười lần nhưng miễn là thực thi mã máy tương đương.*
  - *Việc phát hiện lỗi gõ bởi trình biên dịch là khó khăn vì bất kỳ biến có thể được gán một giá trị thuộc bất kỳ kiểu nào.*

1.9 Lifetime :

- Thời gian tồn tại của một biến là thời gian mà nóđược kết hợp với một ô nhớ cụ thể .
- Các loại biến số theo thời gian tồn tại:
  - *Tĩnh.*
  - *Ngăn xếp động .*
  - *Explicit heap dynamic*
  - *Implicit heap dynamic.*

1.9.1 Lifetime biến tĩnh :

- Các biến tĩnh được kết hợp với các ô nhớ trước khi quá trình thực thi bắt đầu và vẫn kết hợp vào cùng một ô nhớ trong suốt quá trình thực thi.

Ví dụ : tất cả các biến trong FORTRAN 77, biến tĩnh trong C

-Ưu điểm :

- *Tất cả các địa chỉ của vars tĩnh có thể trực tiếp. Không phát sinh chi phí thời gian chạy để phân bổ và giao dịch phân bổ vars.*
- *Có các vars giữ nguyên giá trị của chúng trong khoảng thời gian các lần thực thi riêng biệt của chương trình con.*

- Nhược điểm :

- *Không thể chia sẻ bộ nhớ giữa các biến.*

Ví dụ : Nếu hai mảng lớn được sử dụng bởi hai chương trình con, không bao giờ hoạt động đồng thời, chúng không thể chia sẻ cùng một bộ nhớ cho các mảng của chúng.

1.9.2 Lifetime ngăn xếp động :

- Các liên kết lưu trữ được tạo cho các biến khi khai báo chúng các câu lệnh được xây dựng một cách chi tiết, nhưng có kiểu bị ràng buộc tĩnh.
- Việc xây dựng một khai báo như vậy đề cập đến việc phân bổ lưu trữ và quy trình liên kết được chỉ ra bởi khai báo, diễn ra khi thực thi đến mã mà khai báo là đính kèm.
- Các biến trong dynamic-stack được cấp phát từ runtime-stack .Nếu vô hướng, tất cả các thuộc tính ngoại trừ địa chỉ đều bị ràng buộc tĩnh.

-Ưu điểm:

- *Cho phép đệ quy: mỗi bản sao hoạt động của đệ quy từ chương trình con có phiên bản riêng của các biến cục bộ.*
- *Trong trường hợp không có đệ quy,bảo toàn lưu trữ b/c tất cả các chương trình con chia sẻ cùng một không gian bộ nhớ cho các biến trong cục bộ của chúng.*

1.9.3 Explicit heap dynamic :

- Các ô nhớ không tên được cấp phát và định vị bởi chỉ thị rõ ràng "runtime-instructions", được chỉ định bởi lập trình viên, có hiệu lực trong quá trình thực thi.
- Chỉ được tham chiếu thông qua con trỏ hoặc tham chiếu .

-Ưu điểm: cung cấp cho quản lý lưu trữ động
-Nhược điểm :Khó kiểm soát.

1.9.4 Implicit heap dynamic:

- Biến Impliciti heap dynamic chỉ liên kết với bộ nhớ heap khi chúng được gán giá trị. Sự phân bổ và hủy phân bổ được thực hiện bởi các câu lệnh gán.
- Tất cả các thuộc tính của chúng đều bị ràng buộc mỗi khi chúng được gán.

Ví dụ : tất cả các biến trong APL; tất cả các chuỗi và mảng trong Perl và JavaScript.

-Ưu diểm :Linh hoạt
-Nhược điểm :

- Không hiệu quả, vì tất cả đều là các thuộc tính động –
- Mất khả năng phát hiện lỗi

1.10 Phạm vi của biến :

- Phạm vi của một biến là phạm vi của các câu lệnh trong chương trình mà nó có thể nhìn thấy được

-Các trường hợp điển hình:

- *Được khai báo tường minh=> biến cục bộ*
- *Được truyền tường minh cho chương trình con => tham số*
- *Biến toàn cục => hiển thị ở mọi nơi.*
- *Có hai loại phạm vi chính : là phạm vi tĩnh và phạm vi động*

1.10.1 Phạm vi của biến tĩnh:

- Phạm vi hoạt động của nó chỉ được giới hạn trong hàm mà nó được xác định. Có thể được xác định trước thực thi (*ví dụ: tại thời điểm biên dịch*)

-Để kết nối tham chiếu tên với một biến, bạn (hoặc trình biên dịch) phải tìm khai báo.

```cpp
#include <iostream>
using namespace std;

int phep_cong() {
    int c = 1;
    return 0;
}

int main() {
    cout << phep_cong() << endl;
    return 0;
}
```

Ví dụ :Trong hàm phep_cong() của đoạn mã trên, biến c chính là một biến cục bộ của hàm đó. Nó sẽ không thể được gọi trong hàm main().

1.10.2 Blocks:

- Blocks là một phần mã trong đó các biến cục bộ được cấp phát / không được cấp phát tại bắt đầu / kết thúc khối.
- Cung cấp phương pháp tạo phạm vi tĩnh các đơn vị chương trình bên trong

-Được giới thiệu bởi ALGOL 60 và được tìm thấy trong hầu hết các PL.

Ví dụ :

C and C++:

```
for (...)
{
  int index;
  ...
}
```

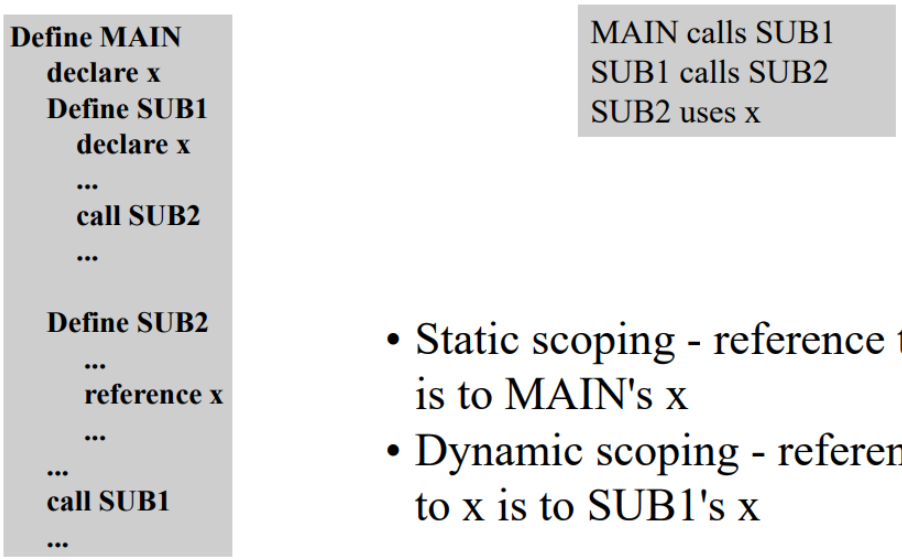
Ada:

```
declare LCL : FLOAT;
begin
...
end
```

1.10.3 Phạm vi của biến động:

- Phạm vi của các biến trong APL, SNOBOL4 và các phiên bản đầu tiên của LISP là phạm vi động.
- Dựa trên việc gọi chuỗi các đơn vị chương trình các tham chiếu đến các biến được kết nối với các khai báo bằng tìm kiếm lại thông qua chuỗi các lệnh gọi chương trình con buộc phải thực hiện đến thời điểm này.

Ví dụ :



1.11 Nguồn tham khảo :

- Type Annotation và Type Inference trong TypeScript là gì?
  Tác giả : Mai Ngọc Trí
  Link
- Are there any variables without a name in programming?
  Tác giả : Joe Zbiciak
  Link
- Chapter 5 : Names, Bindings, Type Checking, and Scopes
  Tác giả : Prof. Dr. Mehmet Alper Tunga
  Link
- Definition of variable
  Tác giả : TechTarget Contributor
  Link

- Varibles
  Link
- Constants, variables and data types
  Link