C++ Training Course

Pointer, Array, String, Structure

Lesson Objectives

- *Understand about pointer and how to use*
- *Understand about array and how to use*
- *Understand about string and how to use*
- *Understand about structure and how to use*

Section 1

Pointer

Pointer. Agenda

- *What is pointer*
- *How to use pointer*
- *Null pointer vs invalid pointer*
- *Pointer arithmetic*
- *How to cast pointer*
- *Pointer vs Reference*
- *Memory layout in C++*
- *Dynamic memory*

Pointer. *What is pointer*

- ***Pointer*** *is a variable whose value is the address of another variable*
- *Size of pointer is same for all data types (4 bytes or 8 bytes)*
- *Declare pointer as form:*

```
type *var_name;
```

```
int *ip;        // pointer to an integer
double *dp;     // pointer to a double
float *fp;      // pointer to a float
char *cp;       // pointer to character
```

Pointer. *How to use pointer*

- *Store address of variable in pointer*

```
int ivar = 20; // actual variable declaration
int *ip = &ivar;    // store address of ivar in pointer variable
```

- *Access value at address in pointer*

```
cout << "Address stored in ip: " << ip << endl;
cout << "Value of *ip: " << *ip << endl;
```

- *Change value at address in pointer*

```
*ip = 30; // at this time, value of ivar is 30
```

Pointer. *Null pointer vs invalid pointer*

- *null pointer* con trỏ null có giá trị 0 (trong C++): hay còn được hiểu là trỏ đến 1 vị trí không hợp lệ
  - ✓ *Constant with a value of zero*
  - ✓ *If pointer is null, it is assumed to point to nothing*

con trỏ null và con trỏ nullptr:
+ con trỏ null trỏ đến giá trị 0, có thể nhầm với giá trị 0 là số nguyên: vd: floo(null) == floo(0)
+ con trỏ nullptr là phiên bản an toàn hơn của null khi nó trỏ đến 1 dữ liệu đặc biệt: std::nullptr_t

- *invalid pointer* con trỏ không hợp lệ
  - ✓ *Point to an invalid memory location*

  (because uninitialized after declare or not set null after delete pointer…)

khi cấp phát động và xóa đi hoặc khi khởi tạo mà không dùng, con trỏ vượt quá giới hạn

Pointer. *Pointer arithmetic*

- *Incrementing & Decrementing (++,--,+,-,+=,-=,...)*

```
int arr[3] = {1, 20, 300};
int *ptr = arr;   // value of *ptr is 1
ptr++;            // value of *ptr is 20
ptr += 1;         // value of *ptr is 300
ptr = ptr - 2;    // value of *ptr is 1
```

1 con trỏ có kích thước bằng 4 hoặc 8 bytes- với mục đích là lưu địa chỉ
+ ở đây con trỏ kiểu int; một int có kích thước là 4 bytes nên khi ppt++ thì sẽ chuyển sang con trỏ tiếp theo (trên thực tế 2 địa chỉ này cách nhau 4 bytes tương ứng với kiểu dữ liệu mà con trỏ trỏ đến)
+tương tự với các kiểu dữ liệu khác

- *Pointer Comparisons (==,<,>,<=,>=,...)*

```
if (ptr == NULL) {
    cout << "null pointer" << endl;
}
while (ptr <= &arr[2]) {
    // TODO
}
```

Pointer. *How to cast pointer*

ép kiểu con trỏ

```
char arr[8] = {1, 2, 3, 4, 5, 6, 7, 8};

char *cp = arr;                    đây là ép kiểu dữ liệu
cout << "value of *cp: " << (int)*cp << endl;          // 1
cout << "value of *(cp + 1): " << (int)*(cp + 1) << endl;  // 2

int *ip = (int *)cp;    đây là ép kiểu con trỏ
cout << "value of *ip: " << *ip << endl;              // 0x04030201
cout << "value of *(ip + 1): " << *(ip + 1) << endl;   // 0x08070605
```


ép kiểu dữ liệu:
+ đây là quá trình chuyển đổi giá trị từ một kiểu dữ liệu này sang một kiểu dữ liệu khác
+ ví dụ chuyển 1 số nguyên thành 1 số thực hoặc ngược lại
+ ép kiểu dữ liệu có thể làm thay đổi biểu diễn của 1 giá trị trong bộ nhớ
+ Cú pháp thường là (type)value, ví dụ (float)myInt hoặc static_cast<float>(myInt) trong C++.

ép kiểu con trỏ:
+ đây là quá trình chuyển đổi kiểu dữ liệu của con trỏ, không phải thay đổi giá trị mà con trỏ trỏ đến
+ ví dụ cùng tại địa chỉ A có thông tin 0xC1200000, nếu là kiểu int thì trình biên dịch hiểu là số 3243393024 nhưng kiểu fload lại là -10.
+ ép kiểu con trỏ không làm thay đổi dữ liệu tại địa chỉ con trỏ trỏ đến mà nó làm thay đổi cách trình biên dịch hiểu dữ liệu đó
+Cú pháp thường là (type*)pointer, ví dụ (char*)myIntPtr hoặc reinterpret_cast<char*>(myIntPtr) trong C++.

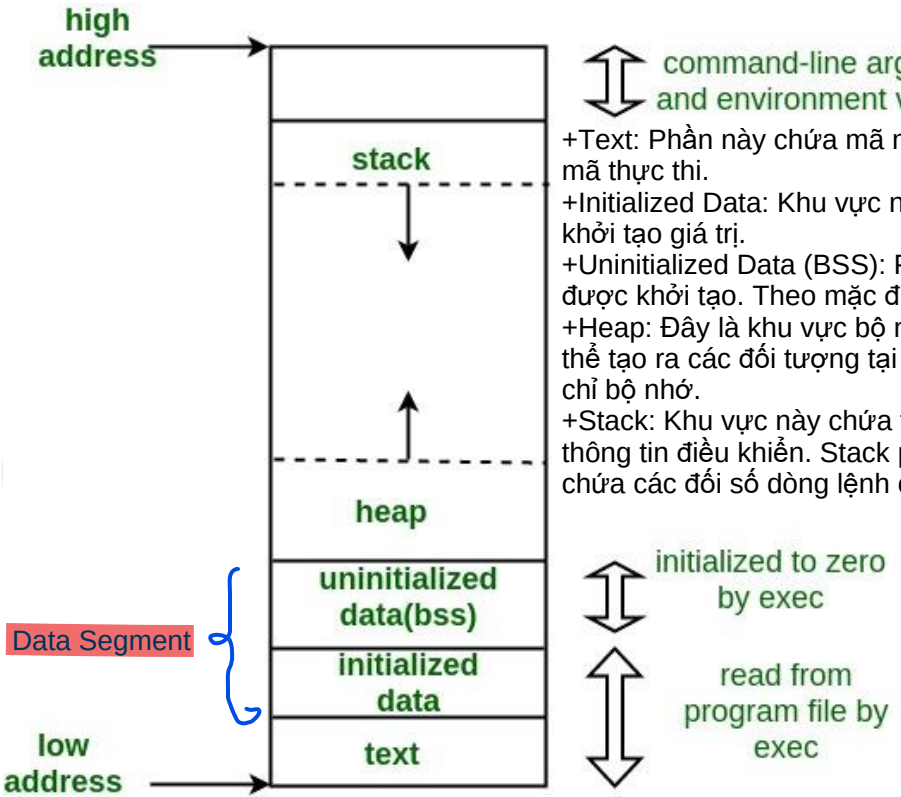
Pointer. *Pointer vs Reference*

Pointer	Reference
Can have null pointer	Cannot have null references
Can be initialized at any time	Must be initialized when created
Can be pointed to another object at any time	Cannot be changed to refer to another object after initialized

reference: tham chiếu: đây là một alias cho một biến đã tồn tại, khi tạo 1 tham chiếu có thể sử dụng tham chiếu này để thay đổi giá trị của biến gốc => đây là cơ sở để sử dụng truyền tham chiếu cho hàm

```
int a = 10;
int& ref = a;
ref = 20; // Thay đổi giá trị của a thông qua ref
std::cout << a; // Output: 20
```

Pointer. *Memory layout in C++*



+Text: Phần này chứa mã máy của chương trình, là nơi lưu trữ các đoạn mã thực thi.
+Initialized Data: Khu vực này chứa các biến toàn cục và static đã được khởi tạo giá trị.
+Uninitialized Data (BSS): Phần này chứa các biến toàn cục và static chưa được khởi tạo. Theo mặc định, chúng được thiết lập là 0 bởi exec.
+Heap: Đây là khu vực bộ nhớ dùng cho việc cấp phát động, nơi bạn có thể tạo ra các đối tượng tại runtime. Heap phát triển theo hướng tăng địa chỉ bộ nhớ.
+Stack: Khu vực này chứa thông tin về các lời gọi hàm, biến cục bộ, và thông tin điều khiển. Stack phát triển theo hướng giảm địa chỉ bộ nhớ và chứa các đối số dòng lệnh cũng như các biến môi trường.

Pointer. *Dynamic memory*

- *new operator*
  - ✓ *Allocate memory dynamically for any data-type (on heap segment)*
  - ✓ *For example*

```
double* pvalue = new double;
```

- *delete operator*
  - ✓ *Free up the memory that the variable occupies*
  - ✓ *For example*

```
delete pvalue;
```

phân biệt new và malloc:
+new: khi khởi tạo tự động xác định kiểu phụ thuộc vào kiểu của dữ liệu => có thể sử dụng để khởi tạo contructor trong class:

```
class MyClass {
public:
    MyClass() { std::cout << "Constructor called\n"; }
    ~MyClass() { std::cout << "Destructor called\n"; }
};
int main() {
    MyClass* obj = new MyClass(); // Gọi hàm khởi tạo
    delete obj;                   // Gọi hàm hủy
    return 0;
}
```

+new: khi khởi tạo có thể khởi tạo luôn giá trị mà cấp phát động trỏ đến:

```
int* ptr = new int(5); // Cấp phát bộ nhớ cho một số nguyên và khởi tạo giá trị là 5
std::cout << *ptr; // Output: 5
delete ptr; // Giải phóng bộ nhớ
```

malloc:
+ không thể sử dụng malloc để khởi tạo làm contructor cho 1 class, khi khởi tạo phải ép kiểu: int* ptr = (int*)malloc(sizeof(int));
+ khi khởi tạo không thể khởi tạo kèm giá trị mà cấp phát động trỏ đến mà phải khởi tạo giá trị sao khi cấp phá động

```
int* ptr = (int*)malloc(sizeof(int)); // Cấp phát bộ nhớ cho một số nguyên
std::cout << *ptr; // Output: Giá trị rác không xác định
free(ptr); // Giải phóng bộ nhớ
```

calloc: khởi tạo malloc n phần tử, mỗi phần tử có kích thước size:

```
void* calloc(size_t num, size_t size);
```

Pointer. Summary

- *Pointer is a variable whose value is the address of another variable*
- *Null pointer is constant with a value of zero*
- *Invalid pointer points to an invalid memory location*
- *Reference cannot null, must be initialized and* cannot be changed to refer to another object
- *Dynamic memory is allocated by new, malloc, alloc on Heap segment and need manage carefully to avoid leak memory*

Section 2

Array

Array. Agenda

- *What is array*
- *How to use array*
- *Array vs pointer*
- *Multi-dimensional array*

Array. *What is array*

- ***Array** is a data structure which stores a fixed-size sequential collection of elements of the same type*

mảng là một cấu trúc dữ liệu nơi lưu trữ một tập hợp tuần tự có kích thước cố định của các phần tử cùng loại

- *All arrays consist of contiguous memory locations*
  - ✓ *Lowest address: first element*
  - ✓ *Highest address: last element*

tất cả các mảng đều bao gồm các vị trí bộ nhớ liên tục:
+ địa chỉ thấp nhất: phần tử đầu tiên
+ địa chỉ cao nhất: phần tử cuối cùng

Array. *How to use array*

- *Declaring Arrays*

```
double balance[10];
```

- *Initializing Arrays*

```
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

- *Accessing Array Elements*

```
double salary = balance[9];
balance[4] = 50.0;
```

Array. *Array vs pointer*

- *Pointers and arrays are interchangeable in many cases. A pointer that points to the beginning of an array can access that array by using either pointer arithmetic or array-style indexing*

```
int arr[3] = {10, 100, 200};
int *ptr = arr;
cout << *(ptr + 1) << endl;  // value of element at index 1
cout << ptr[1] << endl;      // value of element at index 1
```

con trỏ và mảng có thể thay thế cho nhau trong nhiều trường hợp. Một con trỏ chỉ đến phần đầu của một mảng có thể truy cập vào mảng đó bằng cách sử dụng cả toán tử số học con trỏ và cách chỉ của mảng

- *Pointers and arrays are not completely interchangeable because array is a constant that points to beginning of array and cannot change*

```
*arr = 1;        // correct syntax
*(arr + 1) = 2;  // correct syntax
arr++;           // incorrect syntax
```

Con trỏ và mảng không thể hoàn toàn thay thế cho nhau vì mảng là một hằng số chỉ đến phần đầu của mảng và không thể thay đổi

Array. *Multi-dimensional array*

- *Declaring Arrays*

```
int arr[3][4];
```

- *Initializing Arrays*

```
int a[3][4] = {
  {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
  {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
  {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

- *Accessing Array Elements*

```
int val = a[2][3];
a[2][3] = 100;
```

Array. Summary

- *Array is a data structure which stores a fixed-size sequential collection of elements of the same type*

- *Pointers and arrays are interchangeable in many cases but not completely interchangeable because array is a constant that points to beginning of array and cannot change*

Section 3

String

String. Agenda

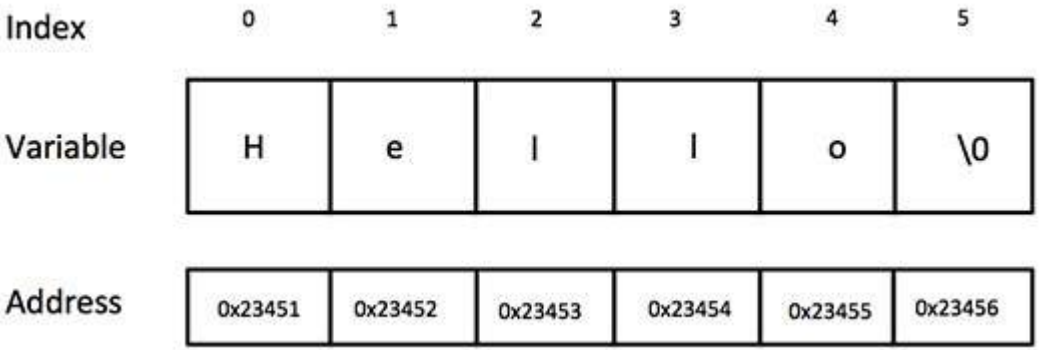
- *What is string*
- *How to use string functions*
- *How to use string class*

String. *What is string*

- ***String*** *is actually a one-dimensional array of characters which is terminated by a null character '\0'*
- *Example: declare string "Hello"*

thực ra chuỗi là 1 mảng một chiều các ký tự kết thúc bằng một ký tự null

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
char greeting[] = "Hello";
```



String. *How to use string functions*

Function	Purpose
strcpy(s1, s2);	Copies string s2 into string s1
strcat(s1, s2);	Concatenates string s2 onto the end of string s1
strlen(s1);	Returns the length of string s1
strcmp(s1, s2);	Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2
strchr(s1, ch);	Returns a pointer to the first occurrence of character ch in string s1
strstr(s1, s2);	Returns a pointer to the first occurrence of string s2 in string s1

vị trí xuất hiện đầu tiên

String. *How to use string class*

- *The standard C++ library provides a string class type that supports all the operations mentioned above, additionally much more functionality*

Thư viện chuẩn C++ cung cấp một loại lớp chuỗi (string class) hỗ trợ tất cả các thao tác được đề cập ở trên, bên cạnh đó còn cung cấp nhiều chức năng phong phú hơn nữa

```
string str1 = "Hello";
string str2 = "World";

string str3 = str1;      // copy str1 into str3
string str4 = str1 + str2;  // concatenate str1 and str2
int len = str4.size();      // get lenght of str4
```

String. Summary

- *String is actually a one-dimensional array of characters which is terminated by a null character '\0'*
- *C++ supports a wide range of functions that manipulate null-terminated strings*
- *The standard C++ library provides a string class type that supports all the operations mentioned above, additionally much more functionality*

Section 4

Structure

Structure. Agenda

- *What is structure*
- *How to use structure*
- *How to use pointer to structure*
- *The keyword typedef in C++*

Structure. *What is structure*

- ***Structure** is user defined data type which allows you to combine data items of different kinds*
- *Structure are used to represent a record with different information*
- *Example: you want to keep track of your books in a library with the following attributes*
  - ✓ *Title*
  - ✓ *Author*
  - ✓ *Subject*
  - ✓ *Book ID*

+Cấu trúc là một kiểu dữ liệu do người dùng tự định nghĩa, cho phép bạn kết hợp các mục dữ liệu khác nhau.
+Cấu trúc được sử dụng để đại diện cho một bản ghi với các thông tin khác nhau.

Structure. *How to use structure*

- *Defining a Structure*

```
struct Books {
   int   book_id;
   char  title[50];
   char  author[50];
};
```

- *Accessing Structure Members*

```
struct Books book1;
book1.book_id = 1;
strcpy(book1.title, "Learn C++ Programming");
strcpy(book1.author, "Chand Miyan");
```

Structure. *How to use pointer to structure*

- *Declare pointer to structure*

```
struct Books *pbook = &book1;
```

- *Accessing Structure Members*

```
cout << "Book title : " << pbook->title << endl;
```

Structure. *The keyword typedef in C++*

- *There is an easier way to define structures or you could "alias" types you create*

```
typedef struct {
   int   book_id;
   char  title[50];
   char  author[50];
} Books;
```

- *You can use Books directly to define variables without using struct keyword*

```
Books book2;
```

Structure. Summary

- *Structure is user defined data type which allows you to combine data items of different kinds*
- *Use '.' operator to access member of structure*
- *Use '->' operator to access member of structure pointer*

References

- https://www.tutorialspoint.com/cplusplus/cpp_pointers.htm
- https://www.tutorialspoint.com/cplusplus/cpp_dynamic_memory.htm
- https://www.tutorialspoint.com/cplusplus/cpp_references.htm
- https://www.tutorialspoint.com/cplusplus/cpp_arrays.htm
- https://www.tutorialspoint.com/cplusplus/cpp_strings.htm
- https://www.tutorialspoint.com/cplusplus/cpp_data_structures.htm

Lesson Summary

- *Pointer*
- *Array*
- *String*
- *Structure*

Thank you