Lecture #05

Pointer(2), Dynamic Allocation and Reference

SE271 Object-Oriented Programming (2020)

Yeseong Kim

Original slides from Prof. Shin at DGIST

Short Notice

- Some questions about HW1
  - The skeleton had a minor bug! Thanks for reporting!
  - I will answer other questions by emails and LMS notice

- Good news! How to get bonus points!
  - I will ask a few of simple short quiz during the class
  - For every answer, you can get one point for HW

Today's Topic

- Arrays to Pointers
- Dynamic Memory allocation
  - C style allocation
  - C++ style allocation
- Function pointer
- Reference Type

Arrays to Pointers

- sizeof(X)
  - Return the size of X
  - sizeof(array)

- int ary[13]; array == & ary[0];
  - Then, &ary?

- Read right to left, and () first
  - int(*whatisit)[13] : pointer to array[13] of int
  - int* whatisthat[13] : array[13] of pointer to int

Passing Array to Function

```
void func1(int* ary) {
        cout << "func1" << endl;
        cout << ary << endl;
        cout << sizeof(ary) << endl;
}

void func2(int ary[]) {
        cout << "func2" << endl;
        cout << ary << endl;
        cout << sizeof(ary) << endl;
}
```

- Note: You cannot return “array”

```
void func3(int ary[13]) {
        cout << "func3" << endl;
        cout << ary << endl;
        cout << sizeof(ary) << endl;
}

void func4(int (*ary)[13]) {
        cout << "func4" << endl;
        cout << ary << endl;
        cout << *ary << endl;
        cout << **ary << endl;
        cout << sizeof(ary) << endl;
        cout << sizeof(*ary) << endl;
}
```

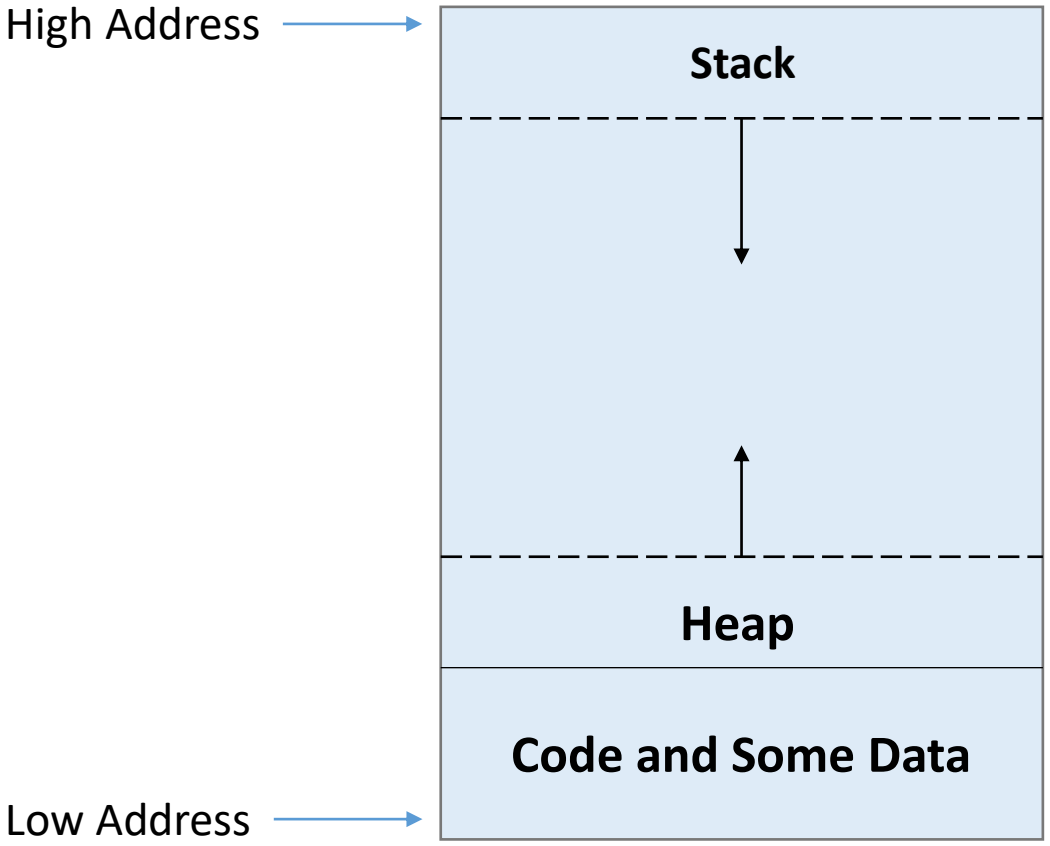
Dynamic Memory Allocation

- **What is it?**
  - C/C++ support allocating/deallocating memory dynamically

- **Why do we need it?**
  - Sometimes, we may not know exact amount of memory before execution
    - Therefore, we need the way to allocate memory dynamically while running
  - Local variables are limited to access
    - within certain block(function)
    - until its life time

Stack, Heap, and Address

- Memory is a tape, and computer saves changeable data in two directions
  - Access the bytes in the tape using "address"



Stack stores the data that
we may know at the *compile-time*

Heap stores the data that
we may know while the program is running

Dynamic Memory Allocation (c style)

- **Syntax**

```
#include <stdlib.h>
void* malloc(size_t size);  // Memory allocation
void free(void * ptr);      // Memory deallocation
```

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
int* ptr;
ptr = (int*)malloc(sizeof(int));
*ptr = 123;
// cout <<   "value of *prt :   " << *ptr;
printf("value of * ptr : %d", *ptr);
free(ptr);
```

```
value of * ptr : 123
```

Dynamic Memory Allocation (C++ style)

- **Syntax**

```
data_type * ptr;
ptr = new data_type;          // Memory allocation
delete ptr;                   // Memory deallocation
```

- **Example**

```
int* ptr;
//ptr = (int*)malloc(sizeof(int));
ptr = new int;
*ptr = 123;
cout << "value of * prt : " << *ptr;
delete ptr;
```

```
value of * ptr : 123
```

Example: Array

- **Single dimensional array**

```
int* ptr;
ptr = new int[10];
// ptr = new int[10] {0};
ptr[0] = 0;
*(ptr + 1) = 1;
for (int i = 0; i < 10; i++)
    *(ptr + i) = i;
for (int i = 0; i < 10; i++)
    cout << ptr[i] << " ";
delete ptr;
delete[] ptr;
```

```
0 1 2 3 4 5 6 7 8 9
```

Memory Leak and Dangling Pointer

- **Memory Leak**

```
int* ptr = new int{ 0 };
int* ptr2 = new int{ 0 };
*ptr = 10;
ptr = ptr2;    // memory pointed by the ptr becomes garbage
```

- **Dangling Pointer**

```
int* ptr = new int{ 0 };
…
delete ptr;
*ptr = 10;     // ???
```

Reference Type

- A **reference** is a type of C++ variable that acts as an **alias** to another
- **Syntax**
  - Declaration

```
data_type * variable_name;
```

  - Initialization (mandatory with delaration)

```
char & name;          // error
int iNum = 10;
int & rNum = iNum;    // int * pNum = &iNum;
```

```
namespace abc {
    int a = 10;
}
int main() {
int& rA = abc::a;
cout << rA;
```

  - Usage

```
cout << "value:" << iNum << " address:" << &iNum << endl;
cout << "value:" << rNum << " address:" << &rNum << endl;
```

```
value:10 address:003BFBE4
value:10 address:003BFBE4
```

Example: call by value, call by reference

```
#include <iostream>
void intSwap1(int  num1, int  num2) {
    int temp{num1};
    num1 = num2;
    num2 = temp; }
void intSwap2(int* num1, int* num2) {
    int temp{*num1};
    *num1 = *num2;
    *num2 = temp;}
void intSwap3(int & num1, int & num2) {
    int temp{num1};
    num1 = num2;
    num2 = temp; }
int * f1(){
    int iNums[3] {1,2,3};
    return iNums; }
```

```
int main(){
    int iNum1{ 1 };
    int iNum2{ 3 };
    std::cout << iNum1 << " " << iNum2 << std::endl;
    intSwap3(iNum1, iNum2);
    std::cout << iNum1 << " " << iNum2 << std::endl;

    int * pNums = f1();
    std::cout << *pNums << std::endl;

    return 0;
}
```

1. Not limited to scope
2. Not copying all data but address

Function Pointer

Syntax

```
// declaration
return_type (* function_pointer) (parameters);
// assignment
function_pointer = function_name;
```

Example

```
int iMenu{ 0 };
int iNum1, iNum2;
int (*func_ptr) (int, int);
cout << "Select Menu (1=add, 2=sub)? ";
cin >> iMenu;
cout << "Left operand: ";
cin >> iNum1;
cout << "Right operand: ";
cin >> iNum2;
func_ptr = (iMenu == 1) ? Add : Sub;
cout <<"Result : " << func_ptr(iNum1, iNum2);
```

```
Select Menu (1=add, 2=sub)? 1
Left operand: 10
Right operand: 3
Result : 13
```

Example: Pointers

- Read right to left, and () first

```
x           -- x
x()         -- is a function
*x()        -- returning a pointer
(*x())[]    -- to an array
*(*x())[]   -- of pointers
(*(*x())[])() -- to functions
char (*(*x())[])(); -- returning char
```

- char (*(*x())[])() : function returning pointer to array[] of pointer to function returning char
- char(*(*x[3])())[5] : array[3] of pointer to function returning pointer to array[5] of char

Source: "The C Programming Languages"

Pointer and Const

```
#include <iostream>
using namespace std;
void PrintName(char * name) {
    cout << "Name : " << name << endl;
    name[0] = '₩0';
}

int main() {
    char myName[10] = "John";
    PrintName(myName);
    cout << "Name : " << myName;
    return 0;
}
```

- `const char * name;`
- `char * const name;`
- `const char * const name;`

References

- Learn c++
  - https://www.learncpp.com/
  - Chapter 6.9 ~ 6.14
  - Chapter 7.8

ANY QUESTIONS?