Basics: From C to C++

Computer Programming for Engineers (DSAF003-42)
Fall, 2021

Practice 3 : Functions

Instructor:

Youngjoong Ko (nlp.skku.edu)

Reference

- conceptually similar to pointer, but is simpler
- Specified by ampersand (&) after type
- Initialization required in declaration

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int val=10;
7       int& ref=val;
8
9       printf("val=%d, ref=%d\n", val, ref);
10
11      val = 20;
12
13      printf("val=%d, ref=%d\n", val, ref);
14
15      return 0;
16  }
```

```
val=10, ref=10
```

```
val=20, ref=20
```

Call by value

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

void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

int main()
{
    int val1 = 10;
    int val2 = 20;

    cout << "[Before swap]" << endl;
    cout << "val1: " << val1 << endl;
    cout << "val2: " << val2 << endl;

    swap(val1, val2);

    cout << "[After swap]" << endl;
    cout << "val1: " << val1 << endl;
    cout << "val2: " << val2 << endl;

    return 0;
}
```

```
[Before swap]
val1: 10
val2: 20
```

```
[After swap]
val1: 10
val2: 20
```

Call by reference (by reference)

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

void swap(int& a, int& b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

int main()
{
    int val1 = 10;
    int val2 = 20;

    cout << "[Before swap]" << endl;
    cout << "val1: " << val1 << endl;
    cout << "val2: " << val2 << endl;

    swap(val1, val2);

    cout << "[After swap]" << endl;
    cout << "val1: " << val1 << endl;
    cout << "val2: " << val2 << endl;

    return 0;
}
```

```
[Before swap]
val1: 10
val2: 20
```

```
[After swap]
val1: 20
val2: 10
```

Call by reference (by pointer)

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

void swap(int* a, int* b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    int val1 = 10;
    int val2 = 20;

    cout << "[Before swap]" << endl;
    cout << "val1: " << val1 << endl;
    cout << "val2: " << val2 << endl;

    swap(&val1, &val2);

    cout << "[After swap]" << endl;
    cout << "val1: " << val1 << endl;
    cout << "val2: " << val2 << endl;

    return 0;
}
```

```
[Before swap]
val1: 10
val2: 20
```

```
[After swap]
val1: 20
val2: 10
```

Exercise 1

- **Create and implement functions below**
  - A function that increases the value of an int variable passed as an argument by 1.

    void increase()
  - A function that inverses the value of an int variable passed as an argument.

    void inverse()
  - Main function that user input an int variable and test above functions.

    int main()

Output example

```
input integer : 25
increased src: 26
inversed src: -26
```

Constant reference parameters

- Reference arguments inherently “dangerous”

  Caller’s data can be changed

- To protect data use const keyword

  Make arguments “read only” by function

  No changes allows inside function body

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

void swap(const int& a, const int& b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

int main()
{
    int val1 = 10;
    int val2 = 20;

    cout << "[Before swap]" << endl;
    cout << "val1: " << val1 << endl;
    cout << "val2: " << val2 << endl;

    swap(val1, val2);
    return 0;
}
```

```
/home/taehun/projects/lab2.cpp:8:7: error: assignment of read-only reference ‘a’
    8 |     a = b;
      |     ~~^~~
/home/taehun/projects/lab2.cpp:9:7: error: assignment of read-only reference ‘b’
    9 |     b = temp;
      |     ~~^~~~~~
```

Mixed parameter lists

- Can combine passing mechanisms
- Order of arguments in list is critical
- void salaryIncrease(int& a, int b, float c);

  arg1 must be integer type, is passed by reference

  arg2 must be integer type, is passed by value

  arg3 must be float type, is passed by float

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

void salaryIncrease(int& a, int b, float c){
    a *= c;
    cout << "increased total salary: " << a*b << endl;
}

int main()
{
    int salary = 10000;
    int workHour = 8;
    cout << "total salary: " << salary*workHour << endl;

    float increaseRate = 1.1;
    salaryIncrease(salary, workHour, increaseRate);

    cout << "current salary: " << salary << endl;

    return 0;
}
```

```
total salary: 80000
increased total salary: 88000
current salary: 11000
```

Overloading

- Same function name, but different function signature
- Overloading allows to use different function type and different input type

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

int add(int a, int b){
    return a+b;
}
float add(double a, double b){
    return a+b;
}
float add(double a, double b, double c){
    return a+b+c;
}

int main(){
    cout << "First function: " << add(10, 20) << endl;
    cout << "Second function: " << add(10.0, 20.0) << endl;
    cout << "Third function: " << add(10.0, 20.0, 30.0) << endl;
}
```

```
First function: 30
Second function: 30
Third function: 60
```

Exercise 2

- **Create change function and fill main code**
  - Change function get an argument or arguments that are passed by reference
  - First change function changes name
  - Second change function changes age
  - Third change function changes age and score

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

void change(){}
void change(){}
void change(){}

int main(){
    string name = "minsu";
    int age = 20;
    double score = 80.0;

    cout << "name: " << name << endl;
    cout << "age: " << age << endl;
    cout << "score: " << score << endl << endl;

    return 0;
}
```

```
name: minsu
age: 20
score: 80

input new name: jinsu
name: jinsu
age: 20
score: 80

input new age: 22
name: jinsu
age: 22
score: 80

input new age and new score: 25 90.0
name: jinsu
age: 25
score: 90
```

for loop

for(initialization expr; boolean expr; update expr)

{

statement 1;

statement 2;

statement 3; …

}

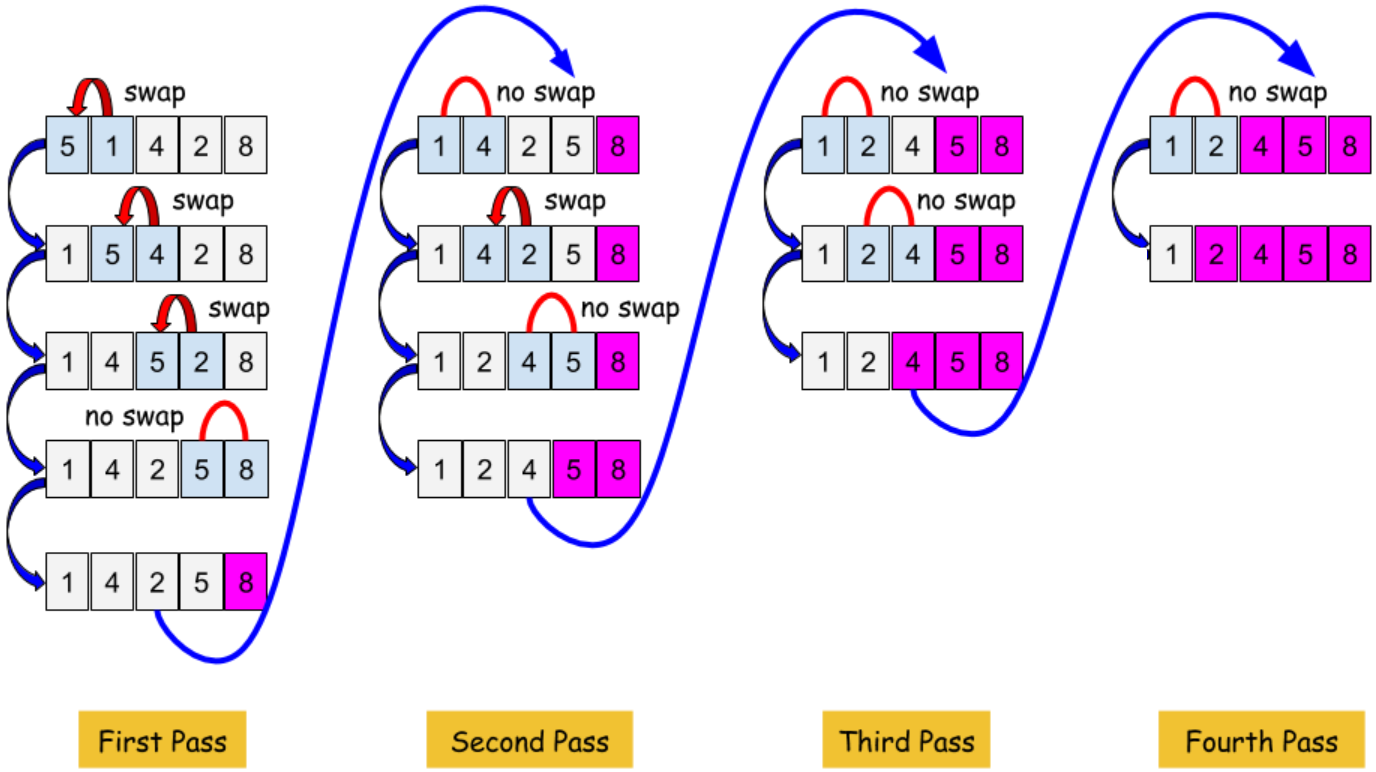
```cpp
#include<iostream>
using namespace std;

int main(){
    int x = 2;
    for (int i =1; i < 10; i++)
    {
        cout << x << " X " << i << "=" << x*i << endl;
    }
}
```

```
2 X 1=2
2 X 2=4
2 X 3=6
2 X 4=8
2 X 5=10
2 X 6=12
2 X 7=14
2 X 8=16
2 X 9=18
```

Assignment

- **Create(Swap, BubbleSort) and implement functions**
  - Bubble sort is a simple sorting algorithm that repeatedly steps through the array, compares adjacent elements and swaps them if they are in the wrong order
  - Double For Loop is recommended



Assignment

- **Create(Swap, BubbleSort) and implement functions**
  - Bubble sort is a simple sorting algorithm that repeatedly steps through the array, compares adjacent elements and swaps them if they are in the wrong order
- **Define main function**
  - All elements of arr are rand()%100
  - Prints all the elements of arr before BubbleSort and after BubbleSort

```
1   #include<iostream>
2   #define SIZE 30
3   using namespace std;
4
5   void Swap(){}
6   void BubbleSort(){}
7
8   int main(void){
9       int seed;
10      cout << "Input Seed: ";
11      cin >> seed;
12      srand(seed);
13      int arr[SIZE];
14
15      BubbleSort(arr,SIZE);
16
17      return 0;
18  }
```

```
Input Seed: 12
Before
60  14  94  8  23  55  29  73  97  64  78  69  57  74  68  4  7  67  87  60  62  48  80  14  38  12  19  66  30  37
After
4  7  8  12  14  14  19  23  29  30  37  38  48  55  57  60  60  62  64  66  67  68  69  73  74  78  80  87  94  97
```

```
Input Seed: 21
Before
56  19  90  22  60  91  63  66  11  55  12  69  61  41  52  99  18  94  66  75  39  55  67  94  33  72  90  74  7  79
After
7  11  12  18  19  22  33  39  41  52  55  55  56  60  61  63  66  66  67  69  72  74  75  79  90  90  91  94  94  99
```