

C++ For C Coders 3

Data Structures
C++ for C Coders

한동대학교 김영섭교수
idebtor@gmail.com

Default function arguments
Reference operator
const, const reference
new and delete operator
command line processing

Default Function Arguments

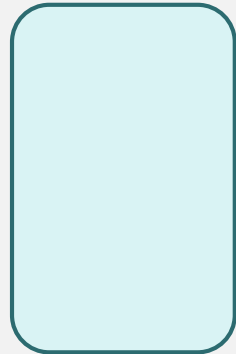
- In calling of the function, if the arguments are not given, default values are used.

```
int exp(int n, int k = 2) {  
    if (k == 2) return (n * n);  
    return (exp(n, k - 1) * n);  
}
```

Default Function Arguments

- In calling a function argument must be given from left to right without skipping any parameter

```
void foo(int i, int j=7);           // right
void goo(int i=3, int j);           //
void hoo(int i, int j=3, int k=7);  //
void moo(int i=1, int j=2, int k=3); //
void noo(int i=2, int j, int k=3);  //
```



Default Function Arguments

- In calling a function argument must be given from left to right without skipping any parameter

```
void foo(int i, int j=7);           // right
void goo(int i=3, int j);           // wrong
void hoo(int i, int j=3, int k=7);  // right
void moo(int i=1, int j=2, int k=3); // right
void noo(int i=2, int j, int k=3);   // wrong
```

Reference Operator &

- A reference allows to declare an alias to another variable.
- As long as the **aliased** variable lives, you can use indifferently the variable or the alias.

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

int main() {
    int x;
    int& foo = x;
    foo = 49;
    cout << x << endl;
    return 0;
}
```

Reference Operator &

- A reference allows to declare an alias to another variable.
- References are **extremely useful** when used with function arguments since it saves the cost of copying parameters into the stack when calling the function.

Reference Operator &

- Swap() in C

```
void swap(_____) {  
    int temp = _____  
    _____  
    _____  
}
```

```
int main() {  
    int i = 3, j = 5;  
    swap(_____);  
    cout << i << " " << j << endl;  
}
```

& is an address operator.

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

```
int main() {  
    int i = 3, j = 5;  
    swap(&i, &j);  
    cout << i << " " << j << endl;  
}
```

Reference Operator &

- Swap() in C++

```
void swap(_____) {  
    int temp = _____  
    _____  
    _____  
}
```

```
int main() {  
    int i = 3, j = 5;  
    swap(_____);  
    cout << i << " " << j << endl;  
}
```

& is a reference operator.

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

```
int main() {  
    int i = 3, j = 5;  
    swap(i, j);  
    cout << i << " " << j << endl;  
}
```


Overloading

- Function overloading refers to the possibility of creating multiple functions with the same name as long as they have different parameters (type and/or number) which is called a signature of function.

C

```
int main() {  
    int i = 3, j = 5;  
    swap(&i, &j);  
    cout << i << " " << j << endl;  
}
```

No Overloading in C

```
int main() {  
    double i = 3, j = 5;  
    swap(&i, &j);  
    cout << i << " " << j << endl;  
}
```

C++

```
int main() {  
    int i = 3, j = 5;  
    swap(i, j);  
    cout << i << " " << j << endl;  
}
```

```
int main() {  
    double i = 3, j = 5;  
    swap(i, j);  
    cout << i << " " << j << endl;  
}
```

Overloading

- Function overloading refers to the possibility of creating multiple functions with the same name as long as they have different parameters (type and/or number) which is called a signature of function.

C++

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

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

C++

```
int main() {  
    int i = 3, j = 5;  
    swap(i, j);  
    cout << i << " " << j << endl;  
}
```

```
int main() {  
    double i = 3, j = 5;  
    swap(i, j);  
    cout << i << " " << j << endl;  
}
```

const Reference

- To prevent the function from changing the parameter accidentally, we pass the argument as **constant reference** to the function.

```
struct Person {  
    char name[40];  
    int age;  
};  
void print(Person k){  
    cout << "Name: " << k.name << endl;  
    cout << "Age: " << k.age << endl;  
}  
  
int main(){  
    Person man{"Adam", 316};  
    print(man);  
    return 0;  
}
```

← C style coding in C++

const Reference

- To prevent the function from changing the parameter accidentally, we pass the argument as **constant reference** to the function.

```
struct Person {  
    char name[40];  
    int age;  
};  
  
void print(const Person& k) {  
    cout << "Name: " << k.name << endl;  
    cout << "Age: " << k.age << endl;  
}  
  
int main(){  
    Person man{"Adam", 316};  
    print(man);  
    return 0;  
}
```

← C style coding in C++

← k is constant reference parameter

What is good about passing by const reference?

- Instead of 44 bytes, only 4 bytes (address) are sent to the function.
- Calling function knows that Person k would not be changed.

Return by reference

- By default in C++, when a function returns a value, it is copied into stack. The calling function reads this value from stack and copies it into its variables.
- An alternative to “return by value” is “return by reference”, in which the value returned is not copied into stack.
- One result of using “return by reference” is that the function which returns a parameter by reference **can be used on the left side** of an assignment statement.

Return by reference

- Modify the following programs such that it sets the maximum element to zero.

```
int max(int a[], int n) {  
    int x = 0;  
    for (int i = 0 ; i < n; i++)  
        if (a[i] > a[x]) x = i;  
    return a[x];  
}  
  
int main() {  
    int a[] = {12, 42, 33, 99, 63};  
    int n = 5;  
  
    for (int i = 0; i < n; i++)  
        cout << a[i] << " ";  
}
```

12 42 33 0 63

Return by reference

- Modify the following programs such that it sets the maximum element to zero.

```
int& max(int a[], int n) {    // returns an integer reference of the max element
    int x = 0;
    for (int i = 0; i < n; i++)
        if (a[i] > a[x]) x = i;
    return a[x];
}

int main() {
    int a[] = {12, 42 , 33 , 99, 63};
    int n = 5;
    max(array, 5) = 0;        // overwrite the max element with 0
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
}
```

12 42 33 0 63

Never return a local variable by reference

- Since a function that uses “return by reference” returns an actual memory address, it is important that the variable in this memory location remain in existence after the function returns.

```
int& foo() {  
    int i;  
    ...  
    return i;                // ERROR! does not exist anymore
```

- Local variables can be return by their values

```
int foo() {  
    int i;                    // Local variable, created in stack  
    ...  
    return i;                 // OK! does not exist anymore
```


malloc & free vs new & delete

- In C, dynamic memory allocation is done with malloc() and free().
- The C++ new and delete operators performs dynamic memory allocation.

```
int *p = (int *)malloc(sizeof(int) * N);  
  
for (int i = 0; i < N; i++)  
    p[i] = i;  
  
free(p);
```

```
int *p = new int[N];  
  
for (int i = 0; i < N; i++)  
    p[i] = i;  
  
delete[] p;
```

Using new & delete

- The **new** operator allocates memory, and **delete** frees it.

```
int *pi = new int;           // pi points to uninitialized int
int *pi = new int(7);        // which pi points has value 7
string *ps = new string("hello"); // ps points "hello"

int *pia = new int[7];       // block of seven uninitialized ints
int *pia = new int[7]();     // block of seven ints values initialized to 0

string *psa = new string[5]; // block of 5 empty strings
string *psa = new string[5](); // block of 5 empty strings
int *pia = new int[5]{0, 1, 2, 3, 4}; // block of 5 ints initialized
string *psa = new string[2]{"a", "the"}; // block of 2 strings initialized
delete pi;
delete[] pia;
```

Command line processing

- Open Atom editor
 - Filename: **args.cpp**
 - Add the source code.
 - Save the file.
- Compile and Execute

```
$ g++ args.cpp -o args
$ ./args Why not change the world?
```
- Read more about this in github:
/nowic/ArgcArgv.md
- Write a function **args_to_strs()** that returns an array of strings to replace both argc and argv;
 - **Use vector<string>**
- Once you complete it, move the function into a new file called **args_to.cpp**.
- Test **args.cpp** and **args_to.cpp**.

```
// args.cpp
#include <iostream>
using namespace std;

int main(const int argc, char** argv) {
    cout << "You entered: "
         << argc << " arguments:" << endl;

    for (int i = 0; i < argc; ++i)
        cout << argv[i] << endl;
    return 0;
}
```

Multiple Source Files

- If you have multiple files to compile and link, for example,
 - Filename: **args.cpp**
 - Filename: **args_to.cpp**
- Compile and execute

```
$ g++ args.cpp args_to.cpp -o args
$ ./args
```

(3)

C++ For C Coders 3

Data Structures
C++ for C Coders

한동대학교 김영섭교수
idebtor@gmail.com

Default Arguments
Reference Operator
const
new and delete operator
command line processing