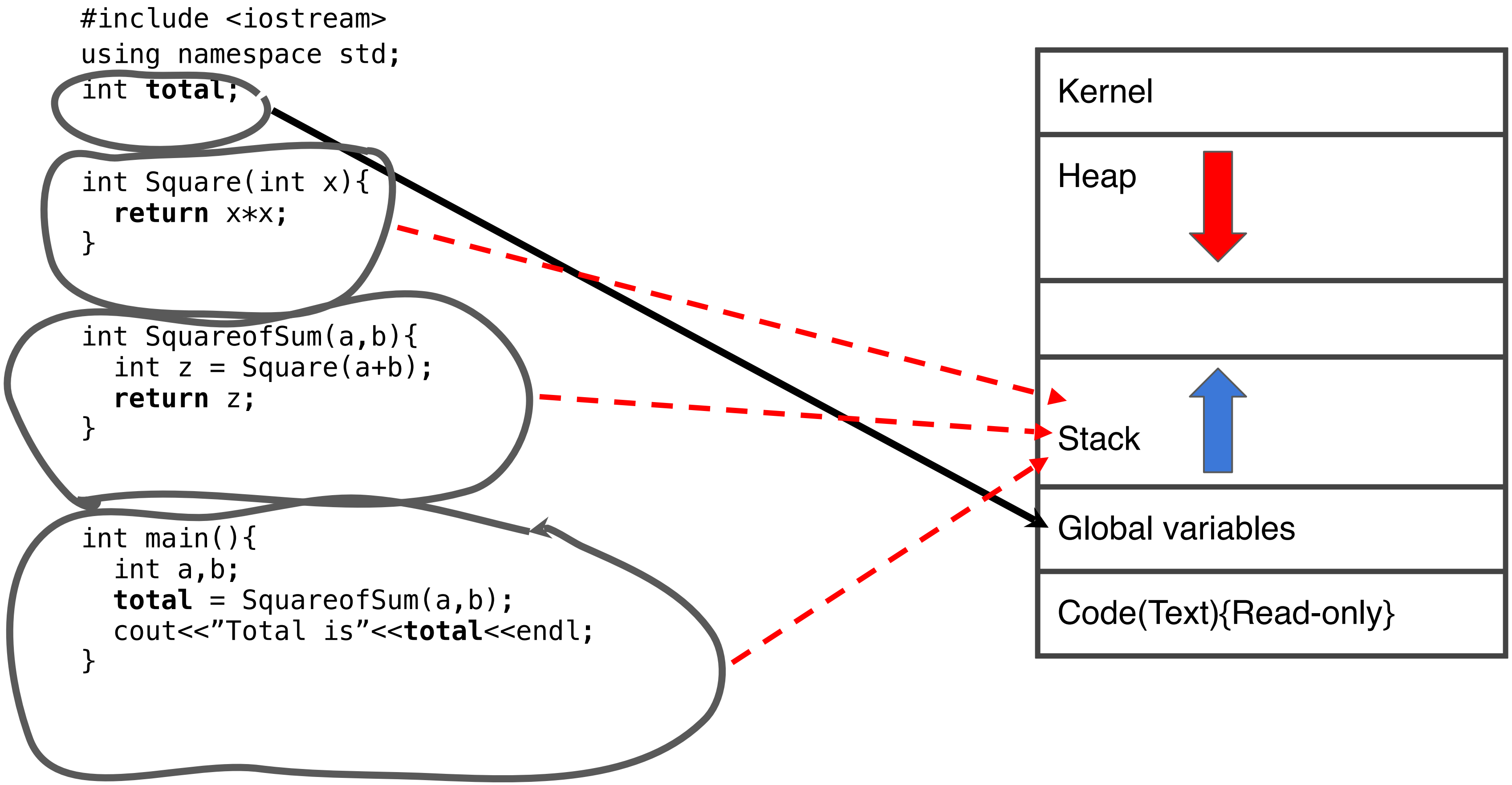
Lecture 3 b

char s[ ] vs char *s



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

int Square(int x){
  return x*x;
}

int SquareofSum(a,b){
  int z = Square(a+b);
  return z;
}

int main(){
  int a,b;
  total = SquareofSum(a,b);
  cout<<”Total is”<<total<<endl;
}
```

Kernel

Heap

Stack

Global variables

Code(Text){Read-only}

```
void mySwap(const char*& a, const char*& b) {
    const char* temp = a;
    a = b;
    b = temp;
}
```

```
char x[4] = "ice";
char y[4] = "the";
mySwap(x, y); //Error
//Memory allocation is in the stack
x[1] = 'f'; y[0] = 's';//valid
```

```
const char* x = "ice";

const char* y = "the";

mySwap(x, y);//No error

//Memory allocation happens in the read
only //text portion

x[1] = 'p'; y[0] = 'q';//Invalid
```

```
char* x = new char[4]{ 'i','c','e','\0' };
char* y = new char[4]{ 't','h','e','\0' };
mySwap(x, y);//No error
//Memory allocation happens in the heap
x[1] = 'a'; y[0] = 's';//valid
```

```
int x[4] = { 1,2,3,4 };
int y[4] = { 2,3,4,5 };
//Memory allocation happens in the stack
mySwap(x, y); //Error
x[1] = 2; //valid
```

```cpp
int* x = new int[3]{ 5,6,7 };
int* y = new int[3]{ 1,2,3 };
//Memory allocation is in the heap
mySwap(x, y);//No error
x[1] = 2; //valid
```

const

A primer

Different types of character array

```
char a[4] = { 'i','c','e','\0' };

char b[] = { 'r','i','c','e','\0' };

char c[] = "nice";

char d[5] = "dice";

char* e = new char[6]{ 's','p','i','c','e','\0' };

const char* f = "price";
```

Printing content of character array

```
cout << a << endl;

cout << b << endl;

cout << c << endl;

cout << d << endl;

cout << e << endl;

cout << f << endl;
```

Printing address of character array

```
cout << (void*)a << endl;

cout << (void*)b << endl;

cout << (void*)c << endl;

cout << (void*)d << endl;

cout << (void*)e << endl;

cout << (void*)f << endl;
```

Printing address of character array

```
//It can also be printed like this:

cout << (void*)a << endl;

cout << (int*)b << endl;

cout << (float*)c << endl;

cout << (double*)d << endl;
```

The thing with void*

```
//It does not allow dereferencing :

int p = 12;

void* x = &p;

cout << *x << endl;//Will give error

int* y = &p;

cout << *x << endl;//Does not give error
```

Printing the content of character array

```
const char* a = “hello”;

cout << &a[1] << endl;//Prints “ello”

cout << &a[2] << endl; //Prints “llo”

cout << (void*)& a[1] << endl; //Prints address of ‘e’

cout << (int*)& a[2] << endl; //Prints address of ‘l’
```

Extra , extra ideas ….

Allocating memory for an array

```
int N = 10;

int arr1[N]; // Will give compile error in Visual Studio

const int M = 10;

int arr2[M];

int arr3[10];//No problem
```

Using the length of a string

```
string str = “hello”;

int len = str.length();

for (int i = 0; i < len; ++i) {

    cout << str[i] << ” ”;

}
```

A more secure way to iterate

```
string str = "hello";

const int len = str.length();

for (int i = 0; i < len; ++i) {

   cout << str[i] << " ";

}
```

Issue with array initialization

```
int* b = new int[2]{ 1,3 }

int* a = new int[2]{ 1,2 };

*(a + 1) = 45;//Valid

a = b; //Valid
```

How to prevent it

Issue with character initialization

```
const char* b = "hello";

const char* a = "jello";

*(a + 1) = 'c';//Invalid, since "hello" is already a const

a = b; //Valid

a = "red" //Valid,"red" is const char*
```

How to prevent it

const char * a and char const * b

```
char* const a = "hello";
//It means that the string literal "hello" should not change
*(a + 1) = 'c'// will give error.
a = "red" // no error,since "red" is itself const char *
```

const char * a and char const * b

```
//const char* and char const* are the same type of identifier
const char* b = "jello";
*(b + 1) = 'a' //will give error
b = "yellow" // no error,since "yellow" is const char*
```

About char *const a

```
char* const a = "red"

//This is in theory should prevent pointer a to be
unmodifiable, ie const.

//However, as “red” is a const char* type, therefore,the
line char* const a, is written as const char* const.

//Therefore if const char* const a = "red", then

*(a + 1) = 't' //Gives error

a = "yellow" //Also gives error
```

But int *const a

```
int* b = new int[2]{ 1,2, };
int* const a = new int[3]{ 1,2,3 };
*(a + 1) = 10;//No error
//But as pointer a is const, therefore:
a = b; //Gives an error
```

const char* const a and char const* const b

```
const char∗ const c = "hello";
∗(c + 1) = 'a';//Wrong, as “hello” is const char∗
c = "tree"; //Wrong , as pointer a is const
```

const char* const a and char const* const b

```
//const char* const can also be written as char const* const;
char const* const  d = "yellow";
*(d + 1) = 'g';//Wrong, gives error
d = "baum";//Also gives error, as pointer d is const
```

const int* const a vs int const* const b

```
//For integer pointer types it translates like this:
const int* const x = new int[2]{ 1,2 }
*(x + 1) = 12; //will be wrong.
x = new int[2]{ 4,5 }; //It will also be wrong
```

const int* const a vs int const* const b

```
//const int* const and int const* const are the same identifier
// works the same way as const int* const
int const* const x = new int[2]{ 1,2 };
```

Summary

```
const char* x = "apple";                  *(x + 1) = 'c'; //Cannot be done
char const* y = "orange";                  x = "football"; //Can be done
```

```
char* const g = "bird";
(The construction is wrong, since "bird" is supposed to be  const char*)
```

```
const char* const a = "ice";               *(x + 1) = 'c';// Cannot be done
char const* const b = "rice";              x = "cricket";// Cannot be done
```

Summary

```
const int* x = new int[2]{ 1,2 }          *(x + 1) = 3 //Wrong

int const* y = new int[2]{ 2,3 }          y = new int[2]{ 1,2 } //Can be done
```

```
int* const z = new int[2]{ 3,4 }        *(z + 1) = 2;//Can be done

z = new int[2]{ 1,2 }//Cannot be done,because ptr is constant
```

```
const int* const a = new int[2]{ 1,2 };      *(a + 1) = 2;// Wrong

int const* const b = new int[2]{ 2,3 };     b = new int[2]{ 1,2 };// Wrong
```