Introduction to Programming (CS 101)

Spring 2024

Lecture 10:

More about references, struct, string data type, recursion (brief introduction)

Instructor: Preethi Jyothi

Based on material developed by Prof. Abhiram Ranade and Prof. Manoj Prabhakaran

Recap (IA)

What is the output of the following program?

```
int change(int a) {
  a += 3;
  return a;
}

main_program {
  int a = 2;
  change(a+2);
  cout << a << endl;
}
```

2

4

7

5

Recap (IB)

What is the output of the following program?

```
int change(int& a) {
  a += 3;
  return a;
}

main_program {
  int a = 2;
  change(a+2);
  cout << a << endl;
}
```

2

4

7

5

None of the above
Compiler Error!

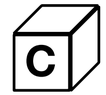
Recap (IC)

What is the output of the following program?

```
int change(int a) {
  a += 3;
  return a;
}

main_program {
  int a = 2;
  change(a+=2);
  cout << a << endl;
}
```

2

4



7

5

Recap (ID)

What is the output of the following program?

```
int change(int& a) {
  a += 3;
  return a;
}

main_program {
  int a = 2;
  change(a+=2);
  cout << a << endl;
}
```

2

4

7

5

Recap (II)

What is the output of the following program?

```
void swp(int& x, int& y) {
   int tmp = x; x = y; y = tmp;
}

void swp(bool& x, bool& y) {
   bool tmp = x; x = y; y = tmp;
}

int main() {
    int a = 1, b = 20;
    bool p = true, q = false;
    swp(a,b); cout << "After swap: " << a << " " << b << endl;
    swp(p,q); cout << "After swap: " << p << " " << q << endl;
}
```

Function overloading: Multiple functions which have the same name, but different input parameter types

Compiler will choose the best match. (Avoid as far as possible.)

output

20 1

0 1

output

Recap (II)

What is the output of the following program?

```
void swp(int& x, int& y) {
   int tmp = x; x = y; y = tmp;
}

void swp(bool& x, bool& y) {
   bool tmp = x; x = y; y = tmp;
}

int main() {
    int a = 1, b = 20;
    bool p = true, q = false;
    swp(a,b); cout << "After swap: " << a << " " << b << endl;
    swp(p,q); cout << "After swap: " << p << " " << q << endl;
}
```

Compile error -- if you had only this definition and try to run `swp(p,q)`. References must match variables **exactly** in data type.

Function overloading

```
void swp(int& x, int& y) {
   int tmp = x; x = y; y = tmp;
}

int swp(int& x, int& y) {
   bool tmp = x; x = y; y = tmp;
}

int main() {
    int a = 1, b = 20;
    bool p = true, q = false;
    swp(a,b); cout << "After swap: " << a << " " << b << endl;
    swp(p,q); cout << "After swap: " << p << " " << q << endl;
}
```

Compile error! Functions that only differ in their return type (`void` vs. `int`, in this case) cannot be overloaded.

Functions and default arguments

CS 101, 2025

Functions and default arguments

- Default arguments cannot be redefined (even if the values are the same)

```
void addten(int a, int b = 10);
void addten(int a, int b = 10) {
  return a + b;
}
```

Compiler Error! `b` was already defined in the declaration. Cannot be redefined in the definition.

- Subsequent declarations (and definition) can extend the default arguments to further parameters

```
int addtens(int a, int b, int c = 10);
int addtens(int a, int b = 10, int c);
int addtens(int a, int b, int c) {
  return a + b + c;
}
```

Word of caution

Is the following code correct?

```
int addtens(int a, int b, int c = 10);
int addtens(int a, int b = 10, int c);
int addtens(int a, int b, int c) {
 return a + b;
}
```

The code you've provided is **incorrect** due to issues with function declarations and default arguments. Let's break it down:

Issues:

1. **Default Arguments and Declaration Order:**
   - The C++ rule for default arguments is that they must be provided from right to left. That means you cannot provide a default for `b` in the second declaration (`int addtens(int a, int b = 10, int c);`) while leaving `c` without a default in that declaration. Instead, the default argument for `c` should also be included.
2. **Inconsistent Declarations:**
   - The declaration `int addtens(int a, int b = 10, int c);` is incorrect because it specifies a default for `b` but not for `c`. This would be invalid in C++ because default arguments must be specified from right to left. If `b` has a default, `c` should also have a default.
3. **Function Definition:**
   - The definition of the function `int addtens(int a, int b, int c)` is missing the expected return type in the code. It should be `int` instead of `void`.

Corrected Code:

cpp Copy

```cpp
int addtens(int a, int b = 10, int c = 10); // Corrected declaration with default a

int addtens(int a, int b, int c) {
    return a + b + c;  // Corrected the function body to return the sum of a, b, ar
}
```

Conclusion:

The original code has a few problems with the ordering of default arguments and missing default values in some declarations. The corrected version ensures that default arguments are specified correctly and that the function definition matches the declaration.

More about references

CS 101, 2025

Returning many values

- Passing by reference can be used as a way to let a function "return" many values
- Can pass placeholders for as many return values as are desired; the function will populate them before returning
- E.g., it is more efficient to compute sine and cosine together

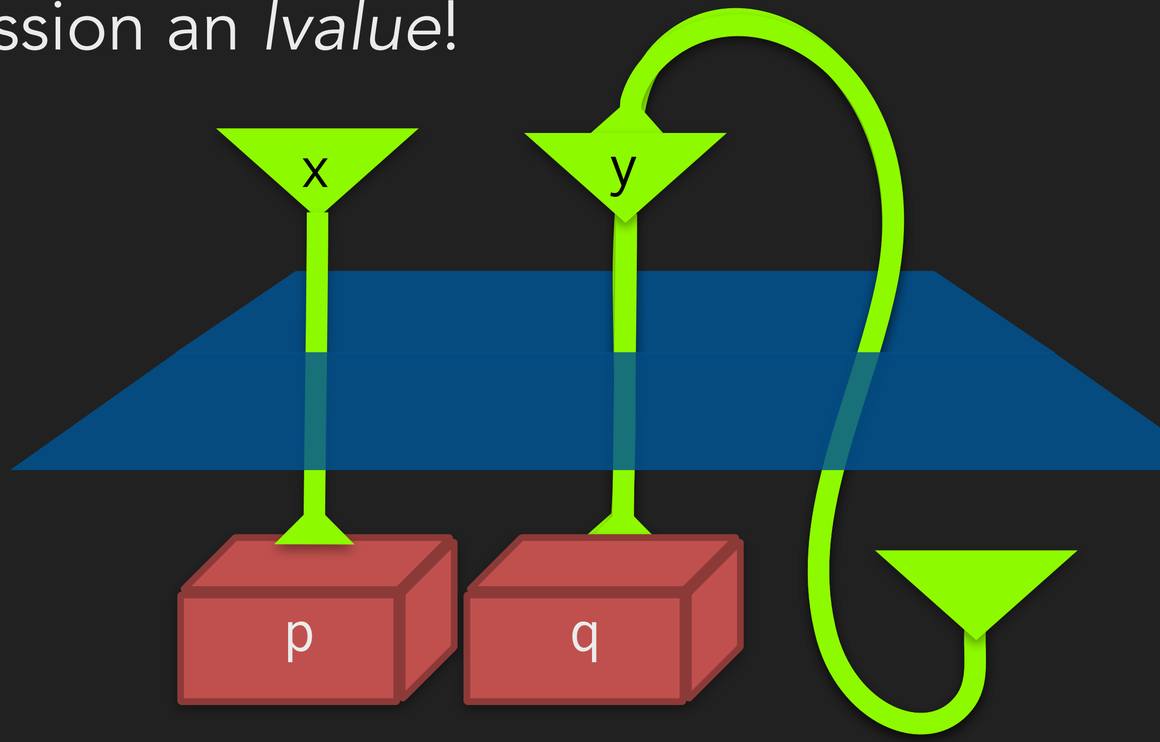
```
void findSinCos(double theta, double& sin, double& cos) {..}

int main() {
  double x, sinx, cosx; cin >> x;
  findSinCos(x,sinx,cosx);
  cout << "sin & cos: " << sinx << ", " << cosx << endl;
}
```

Returning a Reference

- A function can be declared to return a reference too!
- This makes the function evaluation expression an *lvalue*!

```
int& maximum(int& x, int& y) {
  if(x>=y) return x; else return y;
}
```

```
int main() {
  int p, q;
  cin >> p >> q;
  maximum(p,q) = 0;
}
```



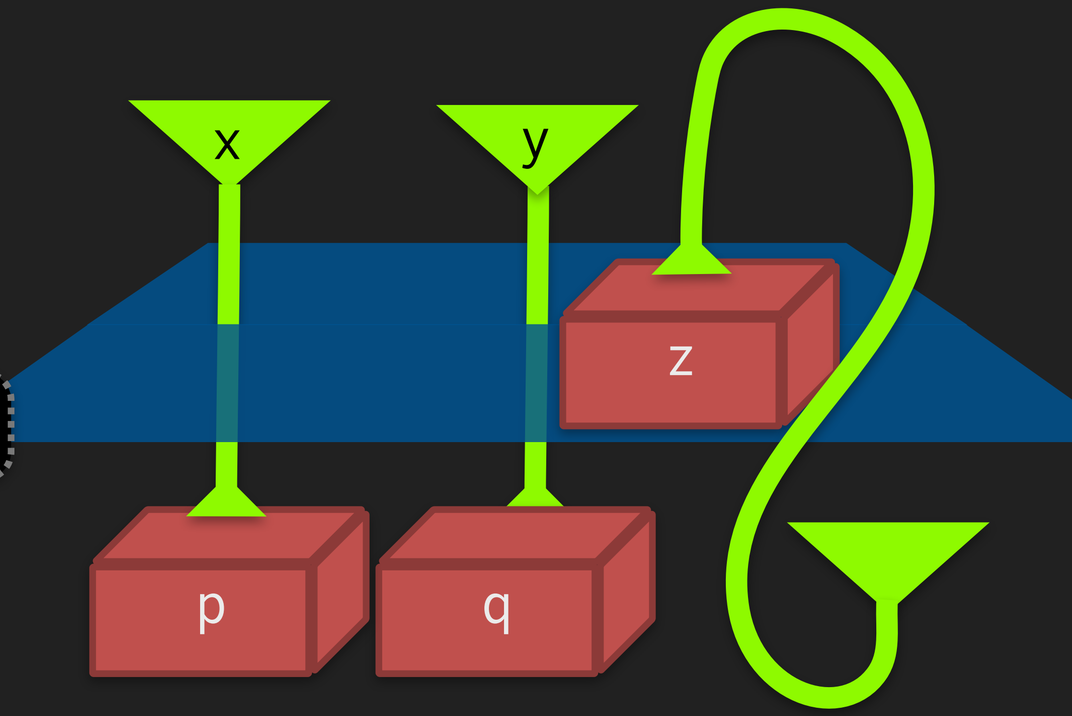
Valid because LHS is an lvalue: a reference to a "box"

Returning a Reference

- Be careful not to return a reference to a local variable
- Compiler can try to warn you

```
int& badMaximum(int& x, int& y) {
  int z = (x>=y)? x: y;
  return z;
}
```

Box for z will be destroyed when the function returns!




```
int main() {
  int p, q;
  cin >> p >> q;
  int& r = badMaximum(p,q);
}
```

Initialising a reference: valid because RHS is an lvalue (box or a tube)

Implement pre-increment operator using references

- Implement `preincr(x)` such that it works like the pre-increment operator `++x`

```
int& preincr(int& m) {
  m = m + 1;
  return m;
}

int main() {
  int x = 1;
  cout << preincr(x) << endl;
  preincr(x);
  cout << preincr(x) << endl;
}
```

Returning a reference

```
int& refMax (int& x, int& y) {
  return (x>y)? x : y;
}

int& refMax2 (int x, int y) {
  return (x>y)? x : y;
}

int& refMax3 (int x, int y) {
   int& z = refMax(x,y);
   return z;
}

main_program {
   int x=1,y=10;
   int& z1 = refMax(x,y);
   int& z2 = refMax2(x,y);
   int& z3 = refMax3(x,y);
}
```

output

z1 will point to y, since it is larger

Bad! Returns reference to a local variable, x or y

Bad! Returns reference z to local variable, x or y

`const` reference

- When passing "big" data, it can be more efficient to pass by reference, because copying data around memory can slow things down
  - But risky: without checking the internals of the function, can't tell if it modifies the argument

- If a function's parameter is a reference, its argument needs an lvalue
  - E.g., `bool isNull(int& x) { return x==0; }`
    `isNull(2); // compiler error`

- Using a `const` reference parameter (`const int&`) solves both these
  - For the second issue: const references can be initialised with rvalues

- Use a `const` reference instead of a reference whenever possible

`const` reference

```
int maxvalue(int& a, int& b) {
  return (a>b) ? a : b;
}

main_program {
  int i = 1;
  cout << maxvalue(1, 2);
}
```

`const` reference

```
int maxvalue(const int& a, const int& b) {
  return (a>b) ? a : b;
}

main_program {
  int i = 1;
  cout << maxvalue(1, 2);
}
```

- One can define `const` references to rvalues such as literals (`1, 2` in the example above) passed as arguments

`const` reference

```
main_program {
   int x;
   cin >> x;
   const int y = 10;
   int &a = max(x, y);
}
```

Compiler error! `max(x, y)` returns a `const` reference to either x or y (whichever is larger). Returning to a reference of type `int` drops the `const` qualifier.

Fix:

```
main_program {
   int x;
   cin >> x;
   const int y = 10;
   const int &a = max(x, y);
}
```

`struct`

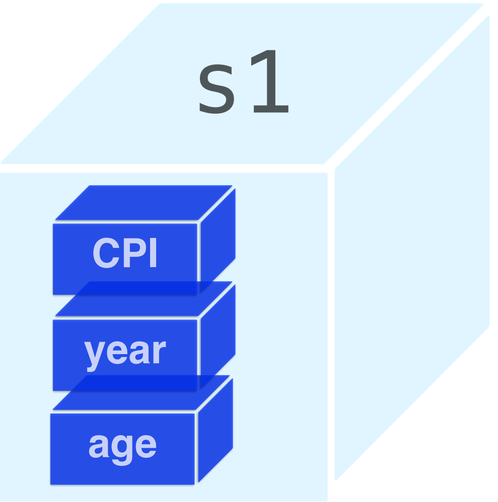
CS 101, 2025

User-defined datatype using `struct`

- So far, we've seen primitive data types such as `bool`, `int`, `float`, etc.
- You can define new data types. Today we'll see how to do it using the `struct` keyword
- Programmer-defined data types can be used similar to built-in ones
  - Define variables of such a data type (they will occupy regions in the memory)
  - Expressions can have values of such a type
  - Can be assigned, passed as arguments (by value or by reference) to functions, and returned by functions (again, by value or by reference)

`struct`

- A struct (for "structure") allows organising several variables into a bundle
  - E.g. for a student, you might need age, degree, CPI, etc.
  - `struct Student { int age, year; float CPI; } ;`
- `struct` can be defined outside of functions
- You can define variables of the new data type. E.g., `Student s1, s2;`
- Think of each `Student` variable as a box with smaller boxes inside them acting as members
  - Can access member variables as `s1.age, s2.CPI`, etc.



struct

- Syntax for assigning values to the variables in a struct

```
struct Student { int age, year; float CPI; } ;
Student s1 = {17, 2024, 9.5}; //all members, in order
Student s2 = {.CPI = 9.3, .age = 18}; //few members, in any order
s2.year = 2023; //access a member individually
```

- Can pass structs to functions (often as references, for efficiency):

```
bool isSameYear(const Student& s1, const Student& s2) {
  return(s1.year == s2.year);
}
```

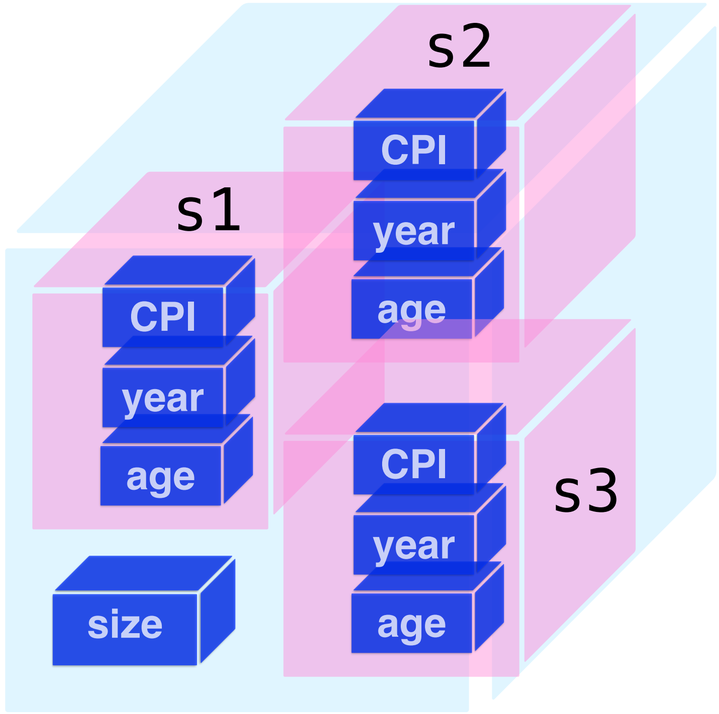
- Functions can return structs too:

```
Student makeCopy(const Student& s1) {
   Student new_stdnt = {s1.age, s1.year, s1.CPI}; return new_stdnt;
}
```

struct

- structs can contain other structs:

```
struct Student { int age, year; float CPI; } ;
struct StudentGrp { Student s1, s2, s3; int size; } ;
```



`string` data type

CS 101, 2025

`string` data type

- C++ strings store sequences of characters

```
string greeting("hello!"); //string greeting = "hello!"; equivalent defn
cout << greeting << endl;
```

- Can use `getline` to take string inputs from users

```
string s;
getline(cin, s); //takes a line of input until newline is encountered
```

- Strings can be passed to and returned from functions

```
void outputString(string s) {
  cout << "The output string is: " << s << endl;
}
```

- Useful functions with `string` type:

```
string s = "hello world"; cout << s.length() << endl; //prints 11
cout << s.find("hello"); //prints 0 i.e. position of the first char
                         //of the substring
string s_new = s + " of C++"; //s_new contains "hello world of C++"
```

Recursion

CS 101, 2025

Recursion

- Recursion is a technique where a function calls itself repeatedly by breaking a problem down into sub-problems.
- The process continues until a *base case* is reached, which stops the recursion and returns a value
- Recursion is terminated on reaching a base case. Double-check that the base case is correct and *will be reached* to prevent infinite recursion

```
void woofs(string s, int cnt) {
  cout << s;
  if(cnt == 3) { cout << "\n"; return; }
  woofs(s, cnt+1);
}

int main() {
  woofs("woof ", 1);
}
```

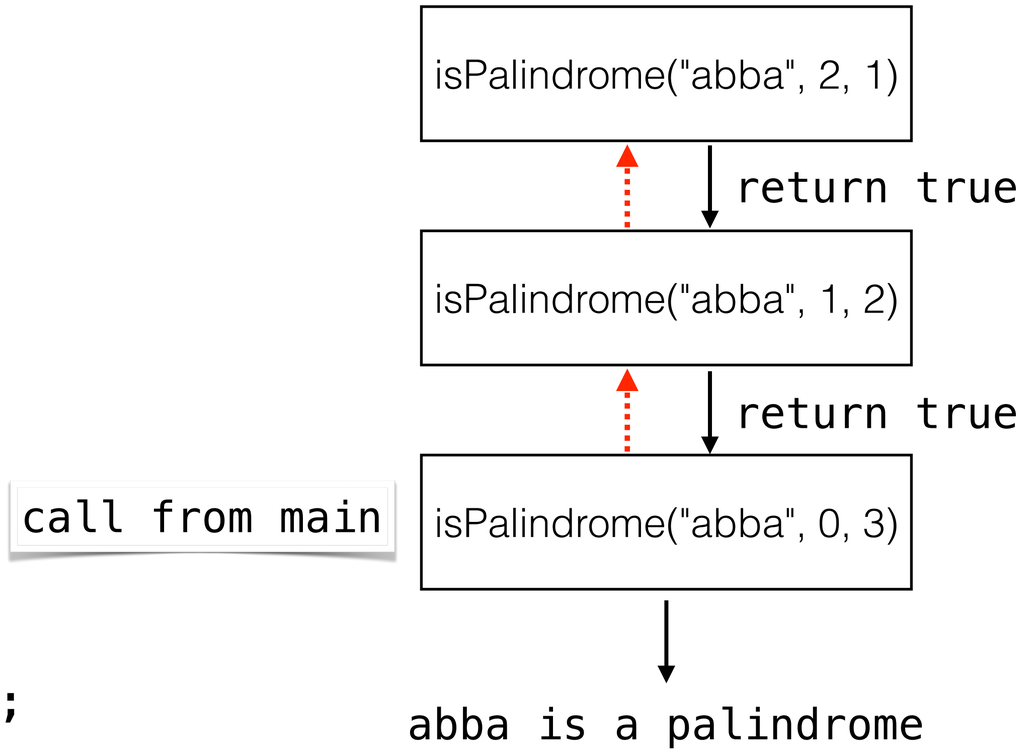
Base case

Check whether a string is a palindrome or not: Use recursion

```
bool isPalindrome(string &s, int start, int end) {
  if (start >= end) return true;
  if (s[start] != s[end]) return false;
  return isPalindrome(s, start + 1, end - 1);
}

int main() {
  string s;
  cin >> s;
  isPalindrome(s, 0, s.length()-1)
              ? cout << s << " is a palindrome\n"
              : cout << s << " is not a palindrome\n";
}
```

Check whether a string is a palindrome or not: Use recursion

```
bool isPalindrome(string &s, int start, int end) {
  if (start >= end) return true;
  if (s[start] != s[end]) return false;
  return isPalindrome(s, start + 1, end - 1);
}

int main() {
  string s;
  cin >> s;
  isPalindrome(s, 0, s.length()-1)
              ? cout << s " is a palindrome\n"
              : cout << s " is not a palindrome\n";
}
```



Next class: More about Recursion

CS 101, 2025