Introduction to Programming (CS 101)

Spring 2024

Lecture 22:

Course conclusion

Instructor: Preethi Jyothi

Recap (I): Constructors/destructors

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

class Atom {
public:
    Atom(const std::string& n) {
        name = n; cout << name << " construct\n";
    }
    ~Atom() { cout << name << " destruct\n"; }
    Atom(const Atom& past) {
        name = past.name;
        cout << name << " copy construct\n";
    }

private:
    string name;
};
```

```
int main() {
    Atom t1("t1");
    Atom t2 = t1;
    Atom* t3 = &t1;
}
```

output

```
t1 construct
t1 copy construct
t1 destruct
t1 destruct
```

Note the pointer `t3` going out of scope does not trigger the destructor.

Recap (II): Copy constructor

Difference between a *shallow copy* and a *deep copy*: What if you comment out the copy constructor definition in blue? This triggers a shallow copy. Both `s1` and `s2` will have pointers to the same memory (reserved for `data` when `s1` was created). When the destructor runs for `s2`, `delete[]` is called twice on the same memory!

```
#include <iostream>
using namespace std;
class Store {
  int size;
  int* data;
  public:
    Store(int s) : size(s) {
      data = new int[size];
      for(int i = 0; i < size; ++i) data[i] = i;
    }
    Store(const Store& m) {
      size = m.size; data = new int[size];
      for(int i = 0; i < size; ++i) data[i] = m.data[i];
    }
    void print() {
      for(int i = 0; i < size; ++i) cout << data[i] << " ";
      cout << endl;
    }
    ~Store() { delete[] data; cout << "destruct\n";}
};
```

output

```
0 1 2
0 1 2
destruct
destruct
```

```
int main() {
  Store s1(3);
  Store s2 = s1;
  s1.print();
  s2.print();
}
```

Two Practice Questions

CS 101, 2025

I - Fill in the blanks

A **Keith number** is defined as a number that appears in the “Keith sequence” associated with that number, as defined below. The Keith sequence of a k-digit number starts with the k numbers which are the digits of the given number; each subsequent number in the sequence is the sum of the previous k numbers in the sequence.

E.g. Consider the 3-digit number 197. Its Keith sequence is 1, 9, 7, 17 (=1+9+7), 33 (=17+7+9), 57 (=33+17+7), 107 (=57+33+17), 197 (=107+57+33), ... Since 197 appears in its Keith sequence, it is a Keith number.

I - Fill in the blanks

```
int main() {
  int n; cin >> n;
  int seq1, seq2, seq3;
  seq1 = n / 100;
  seq2 = BLANK1;
  seq3 = n % 10;
  while (seq3 < n) {
    int seq0 = seq1;
    seq1 = BLANK2;
    seq2 = BLANK3;
    seq3 = BLANK4;
  }
  if (BLANK5) cout << "yes";
  else cout << "no";
}
```

E.g. Consider the 3-digit number 197. Its Keith sequence is 1, 9, 7, 17 (=1+9+7), 33 (=17+7+9), 57 (=33+17+7), 107 (=57+33+17), 197 (=107+57+33), ... Since 197 appears in its Keith sequence, it is a Keith number.

Fill in the blanks in the program to check whether a given input is a 3-digit Keith number or not. Assume the user inputs a positive 3-digit number.

BLANK1: `(n / 10) % 10`

BLANK2: `seq2`

BLANK3: `seq3`

BLANK4: `seq0 + seq1 + seq2`

BLANK5: `seq3 == n`

II - Recursion

Given a positive integer **n**, we want to recursively count the number of ways in which **n** can be partitioned into distinct positive integers. Is the code given below correct? If not, fix it.

Example: n = 5

Answer: 3

The different partitions are:

5

1, 4

2, 3

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

int count(int n, int k) {
  if(n == 0) return 1;
  if(n < 0) return 0;
  return (count(n-k, k+1) + count(n, k+1));
}

int main() {
    int n;
    cin >> n;
    cout << count(n, 1) << endl;
}
```

II - Recursion

Given a positive integer **n**, we want to recursively count the number of ways in which **n** can be partitioned into distinct positive integers. Is the code given below correct? If not, fix it.

Example: n = 5

Answer: 3

The different partitions are:

5

1, 4

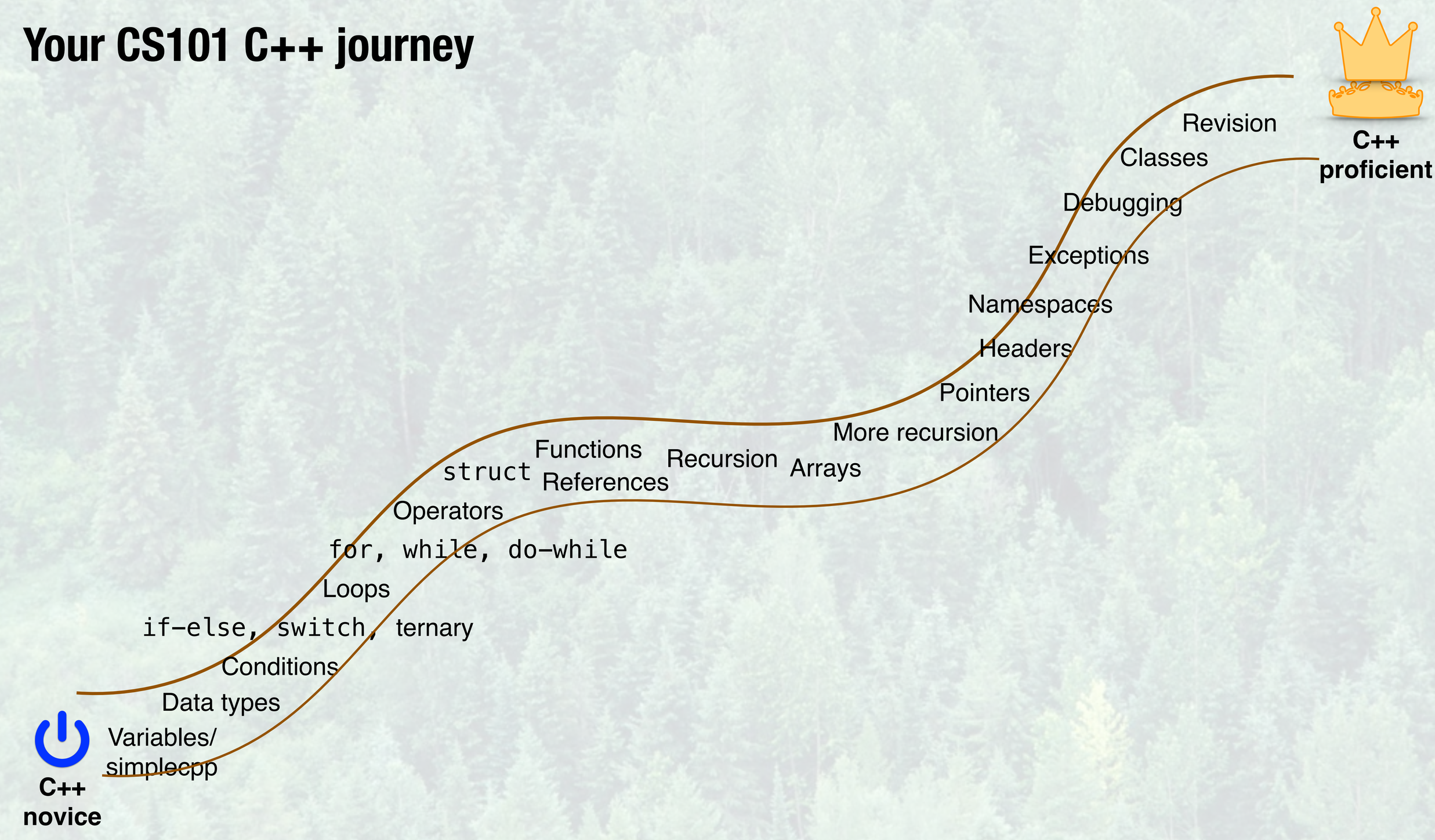
2, 3

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

int count(int n, int k) {
  if(n == 0) return 1;
  if(n < 0 || k > n) return 0;
  return (count(n-k, k+1) + count(n, k+1));
}

int main() {
    int n;
    cin >> n;
    cout << count(n, 1) << endl;
}
```

Course Conclusion

CS 101, 2025

Your CS101 C++ journey



What did you learn from the course?

Recall objectives from lecture 1

- Programming in C++ (Syntax and Semantics)
- Ability to *computationally think* about problems
- Good coding/programming practices
- Learned that seeing programs is necessary but not sufficient

Progression of coding

- Assembly coding to coding using natural language

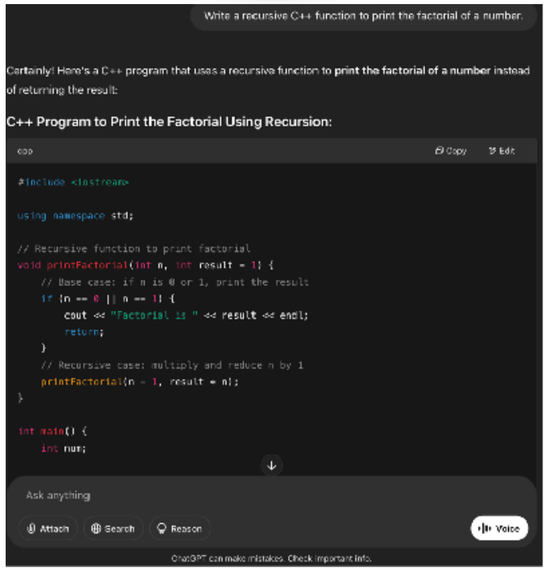
Assembly languages: Low-level

```
_start:
    mov   edx,len      ; length of string
    mov   ecx,msg      ; address of string
    mov   ebx,1        ; file descriptor
    mov   eax,4        ; system call number for write()
    int   0x80         ; system call trap

    mov   ebx,0        ; exit code
    mov   eax,1        ; system call number for exit()
    int   0x80         ; system call trap

section  .data
msg db 'Hello, world!', 0xa
len equ $ - msg
```

High-level languages: (e.g., C++)

```
int main() {
  Store s1(3);
  Store s2 = s1;
  s1.print();
  s2.print();
}
```

Programming using natural language



Why C++?

- Harder to learn but teaches you more transferable skills
- C++ is typically more performant (widely used for native applications)
- Challenges of C++
  - Memory management
  - Bulky syntax
  - Code can get obscure (esp. with preprocessor directives)

Ideas for projects in C++

- Design a social programming platform to support a class like CS101
- Design simple animations for concepts in basic sciences, maths, engineering sciences. E.g., [1]
- Automated stock trading (using financial algorithms)
- Games with a purpose, e.g., ESP game (by Luis Von Ahn [2])
- Word games such as Wordle, crosswords for learning, etc.

---

[1] https://seeing-theory.brown.edu/bayesian-inference/index.html

[2] https://en.wikipedia.org/wiki/ESP_game

Art of writing the right programs

- "*The product is only as good as the plan for the product.*"
  Carefully chart out requirements

- "*The best teamwork is a healthy rivalry.*"
  Developing and testing go hand-in-hand

- "*The database is the software base.*"
  Versioning of code and recording errors.

- "*Don't just fix the mistakes — fix whatever permitted the mistake in the first place.*"
  Make the process extremely detailed.

- Bug-free code is a (near) myth, but we should strive to write (near-)perfect software

https://nparikh.org/assets/pdf/sipa6545/week1-cs-algorithms/write-right-stuff.pdf

Growing presence of AI in software development

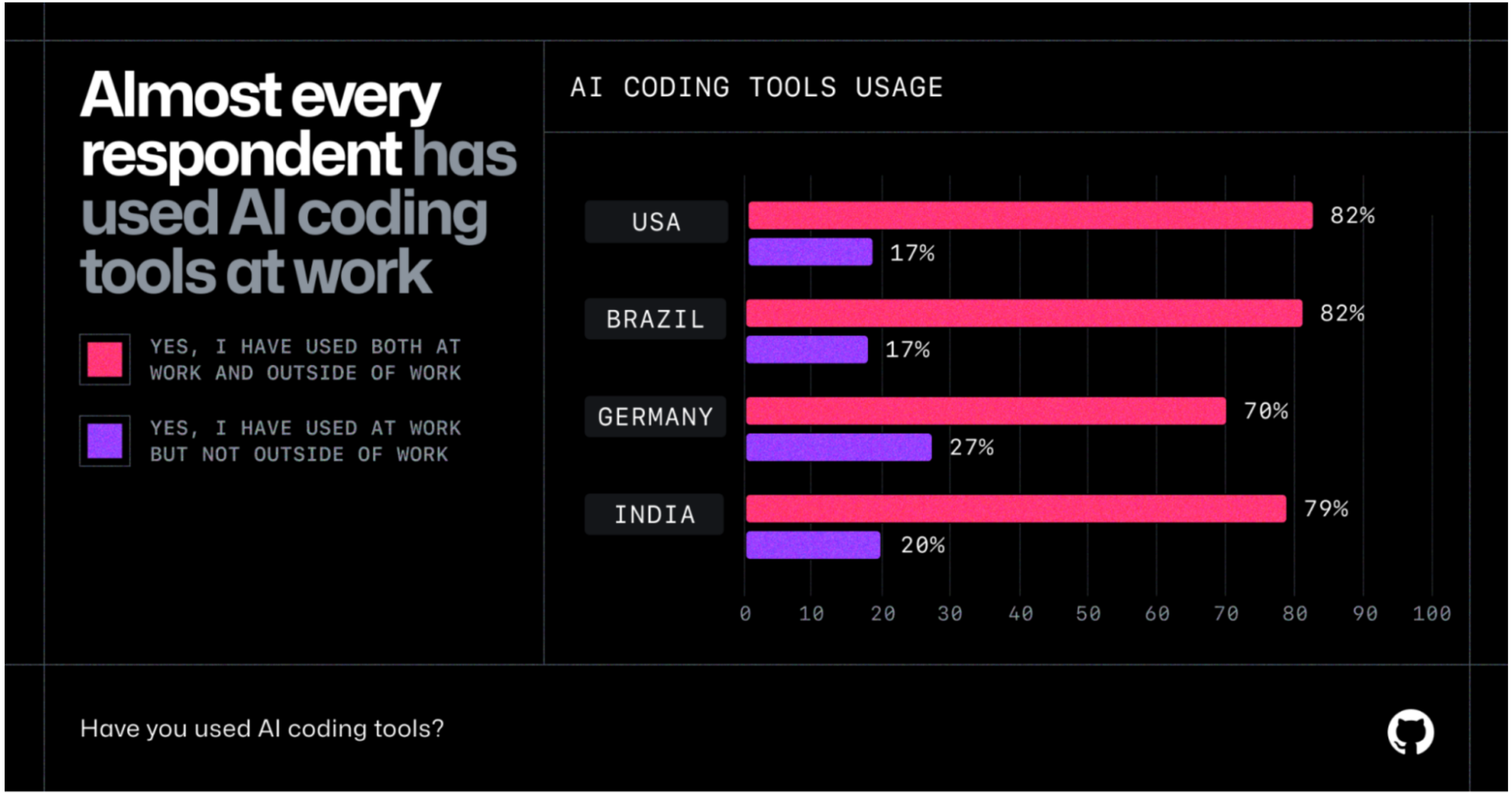


Image from: https://github.blog/news-insights/research/survey-ai-wave-grows/

Using AI for coding

- Not a good idea (as yet!) to use AI as the first resource when you're learning programming
  - Can use it to summarize or rephrase concepts (starting from a trusted source)
- AI coding tools can help you become faster at writing code
- Very useful in automating mundane / low ROI tasks
- Helpful to quickly familiarize yourself with the basics of a new software platform (e.g., Docker) or software tool (e.g., git)

https://nicholas.carlini.com/writing/2024/how-i-use-ai.html

Course Remnants & Concluding Remarks

- Final exam: **April 23, 2025**
  (9 am to 12 pm, Venues: LA001, LA002, LA201, LA202, LC001, LT001
  Syllabus: Lectures 1 to 22 with more emphasis on syllabi post midsem)
- Answer sheets will be shown on or before **May 1, 2025**
- Make-up exams and special test: **May 2, 2025**

- Your programming journey has only begun
- CS101 aims to lay the foundation for you to explore not only C++, but other programming languages (that you will likely use in your years ahead) 🚀