Introduction to Programming

ID1063 • Week 1

Instructor: Rakesh Venkat

July 2026

Today

- Welcome & course logistics
- Why programming
- From source code to a running program
- C: a short history and a first program
- printf, scanf, and your first lab

Course Logistics-1

- Course Code: ID1063
- Course Name: Introduction to Programming
- Instructors: Rakesh Venkat, NR Aravind

- Course slot: F
  - Tue 11am-12noon (lecture, LHC-5),
  - Wed 2.30pm-5.30pm (lab, LHC-5, LHC-6)
  - Fri 10am-11am (lecture, LHC-5)

- Every week will cover a unit of material (typically a new concept).
  - There are 13 weeks in the semester.

Course logistics 2: Moodle

- We will use moodle (moodle.cse.iith.ac.in) for course-related announcements, discussions and to answer queries.
  - Sign up with your IITH email ID.
  - Once signed up, you will be added to the course on Wed.

- Being a large class, use moodle to communicate with TAs and/or the instructor. This avoids missing messages.

Introduction to Programming 2026

Navigation

- My courses
  - Site home
  - Site pages
  - My courses
    - CS6290
  - Courses
    - (hidden)
      - tfb
        - **ID1063-2026**
          - Participants
          - Competencies
          - Grades
          - Logistics and Announcements
          - Lectures
          - Labs
          - Practice Problems
          - Material for Reference
          - Exams
    - Other Departments

Logistics and Announcements

Collapse all

Announcements

Attendance Labs

Lectures

Labs

Practice Problems

Material for Reference

Exams

Course Logistics-3

- Teaching Assistants (TAs) will handle queries, lab sessions, invigilation
  - Head TAs: Subham Bhattacharjee, Mayank V, Abir Banerjee
  - Other TAs : Names will be announced in the second week of the course

- Outside class hours:
  - “I can’t get this to run, please help!”: TAs
  - “I want clarification on a topic/doubt”: Moodle forum, will help others too!
  - “I need instructor permission to …”: Personal message on moodle, followed by office visit if instructor asks you to meet.

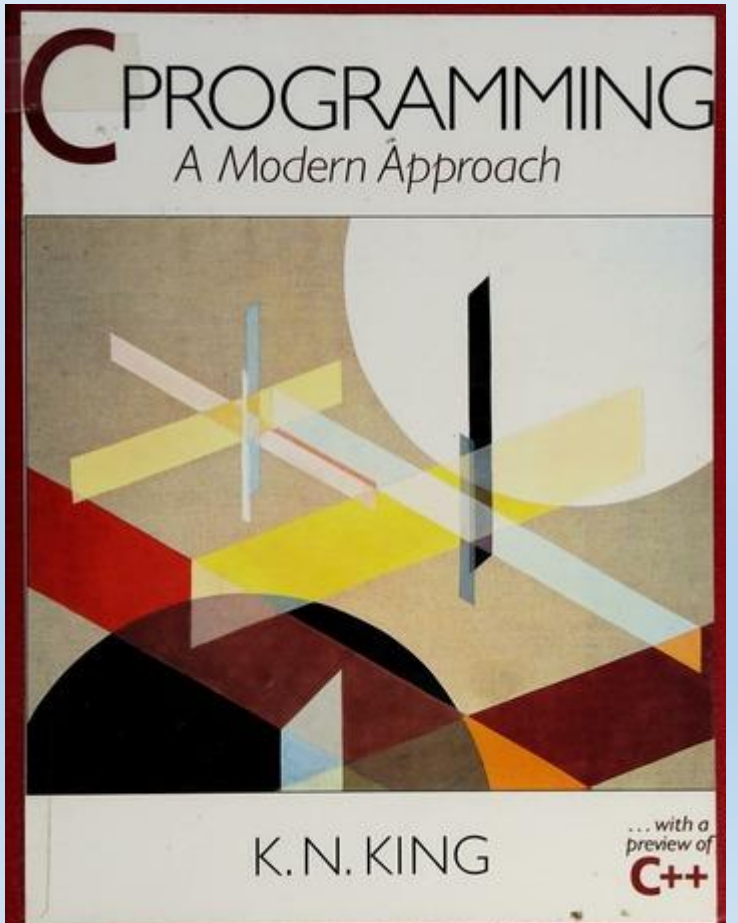
Course logistics 4: Evaluation

- 4 Exams total
  - 2 Theory exams (Multiple choice with negative marking)
  - 2 Lab exams (Done in isolated lab environment without internet access)

Assessment	**Date/window**	**Approximate Weightage (tolerance: 4%)**
Midsem theory	Wed, 16 Sep 2026	22.5%
Midsem lab / Lab Exam 1	Sat, 19 Sep 2026	20%
Endsem theory	Wed, 11th Nov 2026	27.5%
Endsem Lab	Sat, 14th Nov 2026	25%
Lab attendance+submissions	Continual	5%

Course References

- Books.
  - E.g. K.N. King “C Programming A modern approach”



- Other courses (links will be posted)
- Many online resources
  - E.g. W3Schools

Weekly schedule

- Tue Lecture : concepts, demonstrations, questions
- Wed Lab : write, run, test, explain
- Fri Lecture: review/extension

- After class a small amount of regular practice is recommended

- Programming is learnt by doing, not by watching (or asking AI to generate your code for you)

Course survey

A short survey will be shared separately on Moodle.

Questions:

- Prior programming experience
- Access to a laptop for the lab sessions
- OS that you are comfortable using

This is purely informative. We will assume zero prior familiarity with programming.

Learning programming

What this course builds

- Write and test small C programs
- Trace code and diagnose errors
- Use decisions, loops, arrays, strings and functions
- Work with pointers, dynamic memory, structures and files

- Course objective: clear thinking expressed precisely.

What makes programming work

Try a tiny example.
Predict what should happen.
Run it.
Compare, explain, and change one thing.

Confusion is normal; unexplored confusion is the problem.

Why learn programming?

- Turn an idea into an executable process
- Automate repetitive work
- Analyze data and model a system
- Build tools that others can use

Programming is a way to make your reasoning operational in practice.

Why learn it in the age of LLMs?

- An LLM can suggest code; you still decide the problem.
  - You must test whether the result is correct.
  - You must notice missing cases and unsafe assumptions.
  - You must be able to explain and adapt what you submit.
  - You must be able to give implementable specifications for your requirement.

- Programming literacy makes AI assistance more useful and safer.

- Most importantly, learning programming builds a way of careful thinking about problems and identifying potential failures.
  - The thought skills are transferable to other domains.

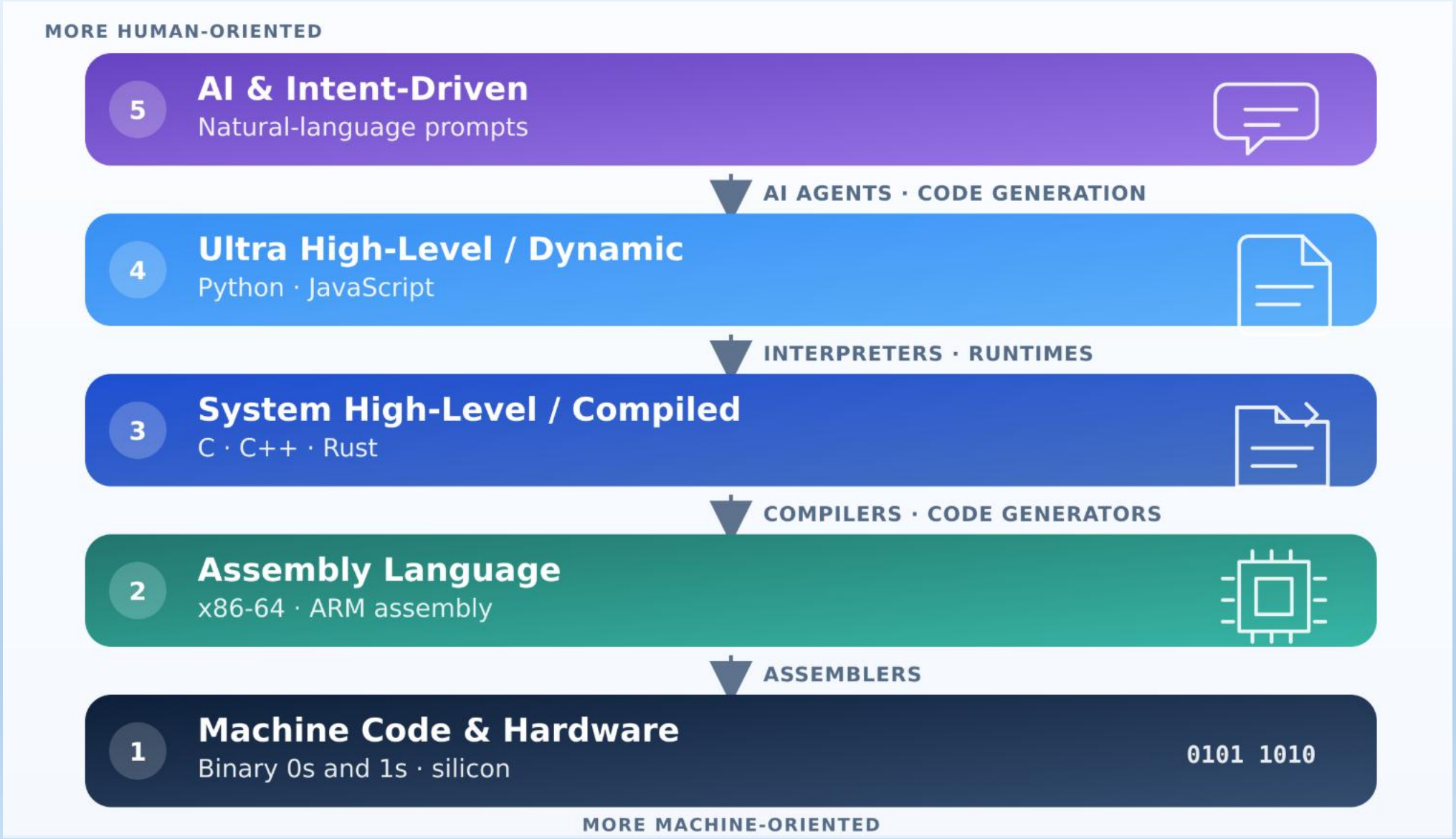
What is programming?

Programming: a precise recipe

- A program is a sequence of instructions.
- The instructions transform **inputs** into **outputs**.
- A computer is extremely fast — and extremely literal.
  - Like a (dumb) genie that does *literally* what it is told to do without deviations

- Our job is to remove ambiguity before the machine runs the recipe.

Programming Abstractions

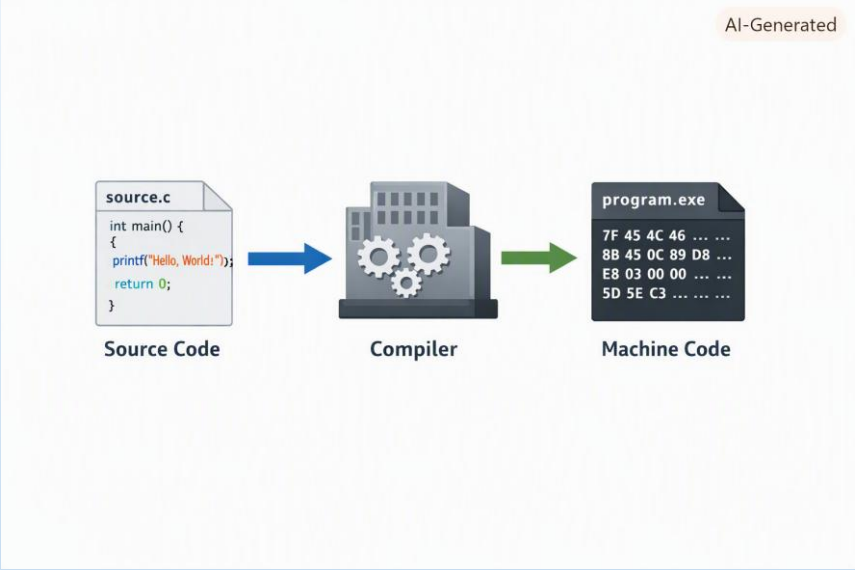


Why not write machine language?

- A processor ultimately executes patterns of bits.
- Those patterns differ across processor families. Too much work to rewrite every program.
- They are difficult for humans to write, ***read and repair***.
- High-level languages let us express the same idea in a human-friendly form.

The programming workflow

- Understand problem → plan → write C source code
- C source code → compiler → executable program
- Executable program + input → output



- The language (C in our case) specifies how to write the instructions (the program).
- The compiler is the translator between our C program and the machine.

- Process is roughly the same for every compiled language.

From a file to a running program

1. Write source code: hello.c
   This is just a readable text file.

2. Compile: gcc hello.c -o hello

3. Run: ./hello (or equivalent)

If the compiler reports an error, read the first message carefully, fix it, and compile again.

gcc == GNU C compiler. More on this later.

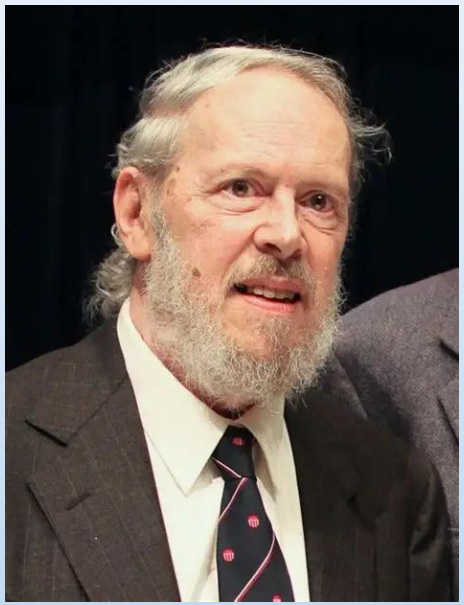
C

C: a brief history

- Developed at Bell Labs by Dennis Ritchie in the early 1970s
- Created alongside the Unix operating system
- Designed to be expressive, efficient and close to the machine

- C became one of the most influential languages in computing.

Contributor	Role	Contribution
Dennis Ritchie	Principal designer	Created C as a systems programming language for Unix; defined its syntax and semantics.
Ken Thompson	Collaborator	Co-developed Unix with Ritchie; influenced C's design through his earlier **B language**, which C evolved from.
Brian Kernighan	Contributor and documenter	Helped popularize C; co-authored *The C Programming Language* book with Ritchie (often called "K&R").
Alan Snyder, Steven C. Johnson, Peter Weinberger, and Mike Lesk	Bell Labs colleagues	Contributed to Unix tools and compilers that used or extended C, helping refine its practical implementation.



Dennis Ritchie (1941-2011)

The 1973 portability story

- Unix was first closely tied to a particular computer.
- Rewriting most of it in C made it practical to move Unix to new machines.
- A powerful idea: write the system once, recompile it elsewhere.

- C was small enough for systems work, but much easier than assembly.

- C keeps evolving
  - Specifications updated time to time (e.g. C89, C99, C11, C17, and C23)
  - Governs syntax, semantics, standard libraries operation

  - We will be using the C17 standard (but most basic stuff is common across standards from C99)

Why C still matters

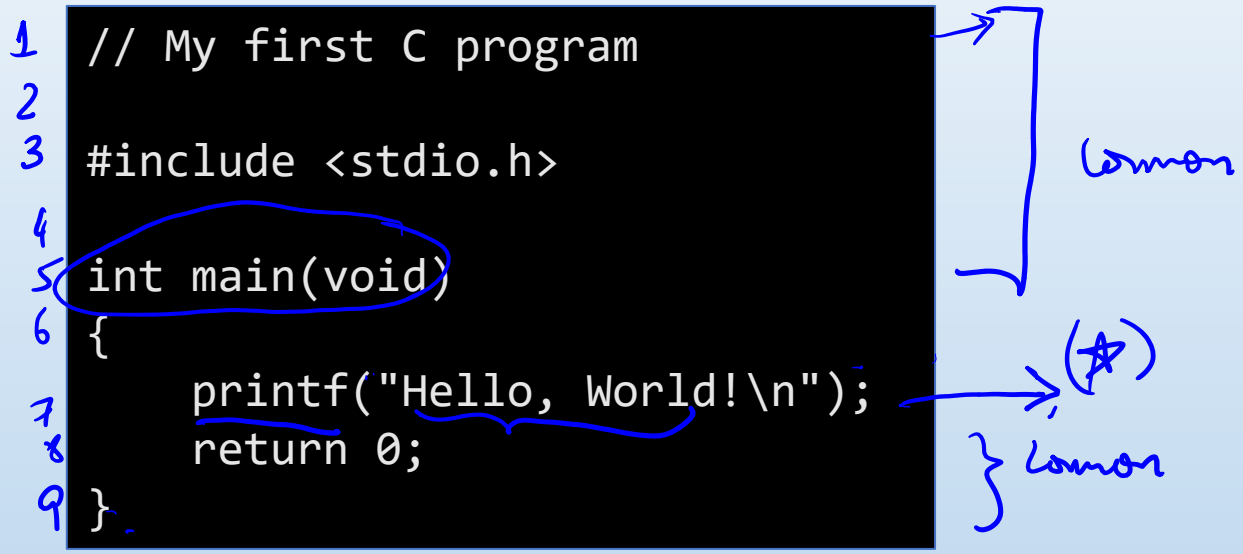
Operating systems and device software
Embedded systems: cars, sensors, appliances and instruments
Networking and performance-critical components

A solid foundation for understanding many newer languages

Programming

Our first programs

Your first C program



Line 1: Comment, ignored by compiler
Line 2: Blank
Line 3: Include standard library to enable input-output
Line 4: Blank
Line 5: The “main function” for the program to execute
Line 6: Start of main function (statements enclosed in brackets)
Line 7: Print statement, printing to standard output
Line 7: returns integer 0
Line 8: End of main function

Compiling and running

```
// My first C program

#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
    return 0;
}
```

1. Store the file using a name
   - E.g. `first-program.c`

2. Ask the C compiler (gcc) to create an *executable* from the file (use a Terminal)

   ```
   gcc first-program.c –o first.exe
   ```

3. Run the created executable file from the Terminal:

   ```
   first.exe
   ```

Comments

- Blank lines
- Comments: use and types
  - Single line comments
  - Multiline Comments
  - Boxed Comments

```
// My first C program : single line
comment

/* Program name: first-program.c
Author: ABC
Updated on: 28 July 2026
*/

#include <stdio.h> //standard library

//Here is the main function
int main(void)
{
    printf("Hello, World!\n");
    return 0;
}
```

Mistakes?

- What can go wrong?
  - Syntax errors
  - Logical Errors

Functions and the main function

- A function is a block of statements, grouped and given a name
- A function usually gives an output
  - Like a mathematical function $f(x) = 2x + 4x^2$
- The function “main” is special, it is always executed first when the program is executed.
  - For the first few weeks, our programs will have only the main function
  - The main function returns a value (0) to the OS signifying that it completed execution, and all is well.

More on `printf`

- Splitting lines: newline is given by `\n`
- Multiple newlines
- Special Characters
- How do you print the exact output:

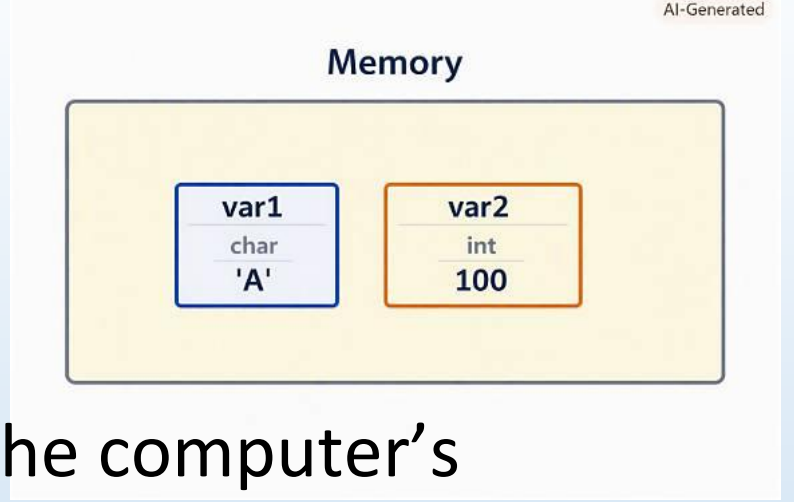
```
The end of the line is denoted by \n.

She said, “You are 100% correct!”.
```

Escape sequences in printf

Escape sequence	Description
\n	Moves the cursor to the beginning of the next line.
\t	Moves the cursor to the next horizontal tab stop.
\a	Produces a sound or visible alert without changing the current cursor position.
\\	Because the backslash has special meaning in a string, \\ is required to insert a backslash character in a string.
\"	Because strings are enclosed in double quotes, \" is required to insert a double-quote character in a string.

Variables

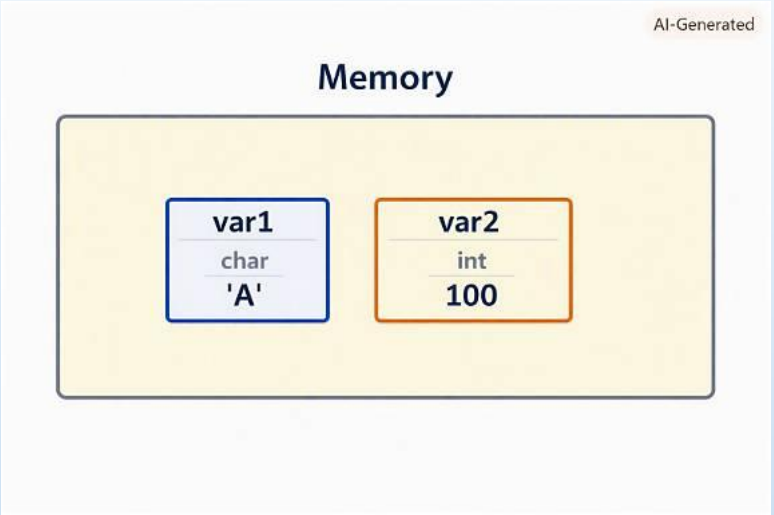


- A variable is a *named storage area* in the computer's memory.

- Every variable is ***declared*** *with a* ***name*** *and* ***type***
  - The type decides the space allocated
  - Examples: Integer, character, floating point numbers

- A variable stores a *value* in the storage that can be updated during the course of the program.
  - Assigning a value is called *assignment*

Example variables



- Integer

  Declaration statements:
  int x;
  int y;
  int z,w;

  Assignment statements:
  x=10;
  y=20;

- Character

  char c;

  c=‘a’

- Double precision floating point

  double f;
  f=3.1415;

Space for Variables

Type	Typical size	Typical range / precision
char	1 byte=8 bits	One character or a small integer. If signed: −128 to 127; if unsigned: 0 to 255.
int	4 bytes	−2,147,483,648 to 2,147,483,647
float	4 bytes	About ±3.4 × 10^38; about 6–7 significant decimal digits of precision
double	8 bytes	About ±1.8 × 10^308; about 15–16 significant decimal digits of precision

Initialization

- Setting the starting value of a variable in the declaration

  ```
  int x=10;
  ```

- A variable which is not initialized is called *uninitialized*