A.

1. ALGOL, developed in mid-1950s, and first appeared in 1958. ALGOL introduced code blocks and the "begin…end" pairs for delimiting them. It was also the first language implementing nested function definitions with lexical scope

2. BASIC, first appeared in 1964. BASIC was one of the few languages that was both high-level enough to be usable by those without training and small enough to fit into the microcomputers of the day.

3. COBOL, first appeared in 1959. COBOL has an English-like syntax, which was designed to be self-documenting and highly readable. It is good for other programmers to understand the codes not written by themselves.

4. C, first appeared in 1972. By design, C provides constructs that map efficiently to typical machine instructions, and therefore it has found lasting use in applications that had formerly been coded in assembly language, including operating systems, as well as various application software for computers ranging from supercomputers to embedded systems.

5. Prolog, first appeared in early 1970s. Prolog is well-suited for specific tasks that benefit from rule-based logical queries such as searching databases, voice control systems, and filling templates.

6. FORTRAN, first appeared in 1957. It is good because it is fast.  It is a popular language in the area of high-performance computing and is a language used for programs that benchmark and rank the world's fastest supercomputers.

B.

Assembly instructions are a direct mapping to opcodes, which are multi-byte values of machine code that can be directly interpreted by the processor. It is quite possible to write a program in opcodes directly by looking them up from a table that lists them with the matching assembly instructions, and hand-determining memory addresses/offsets for things like jumps. The first programs were done by hand-written opcodes.

The first assemblers were written by hand. Those assemblers could then be used to assemble more complicated assemblers, which could then be use to assemble compilers written for higher-level languages, and so on. This process of iteratively writing the tools to simplify the creation of the next set of tools is called bootstrapping.

C.

1. Dennis MacAlistair Ritchie (September 9, 1941 - October 12, 2011) was an American computer scientist.He created the C programming language and, with long-time colleague Ken Thompson, the Unix operating system. He was the "R" in K&R C and commonly known by his username dmr.
2. Alan Mathison Turing (23 June 1912 - 7 June 1954) was a British pioneering computer scientist, mathematician, logician, cryptanalyst and theoretical biologist. Turing is widely considered to be the father of theoretical computer science and artificial intelligence.
3. James Arthur Gosling (born May 19, 1955) is a Canadian computer scientist, best known as the father of the Java programming language. While working towards his doctorate, he wrote a version of Emacs called Gosling Emacs (Gosmacs), and before joining Sun Microsystems he built a multi-processor version of Unix.
4. John Warner Backus (December 3, 1924 - March 17, 2007) was an American computer scientist. He directed the team that invented the first widely used high-level programming language (FORTRAN) and was the inventor of the Backus-Naur form (BNF), a widely used notation to define formal language syntax. He also did research in function-level programming and helped to popularize it.
5. Stephen Gary "Steve" Wozniak (born August 11, 1950), known as "Woz", is an American pioneer of the personal computer revolution of the 1970s. Wozniak is an American inventor, electronics engineer, and computer programmer who single-handedly developed the 1976 Apple I, the computer that launched Apple.
6. Bjarne Stroustrup (born 30 December 1950) is a Danish computer scientist, most notable for the creation and development of the widely used C++ programming language. In 2015, he was made a Fellow of the Computer History Museum for his invention of the C++ programming language.

D.

“goto statement” performs a one-way transfer of control to another line of code. It is bad because it allows you do very weird things, such as jumping into a new scope without leaving the original scope. Also, it would be very difficult for other people to understand your code.

E.

Is it possible to create a function which is able to minimize my codes automatically? Since with Boolean algebra can minimize the gate implementation, can we use similar way that we compile programs to binary codes first, minimize the binary with Boolean algebra, and then transform it back.