Programming

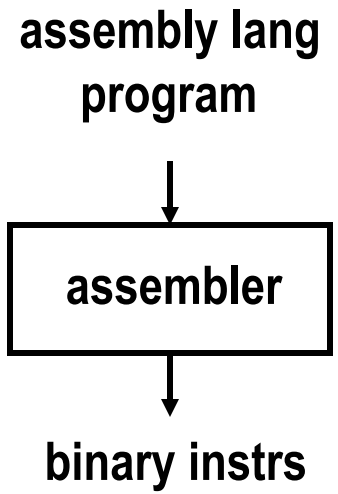
- **it's hard to do the programming to get something done**
- **details are hard to get right, very complicated, finicky**
- **not enough skilled people to do what is needed**
- **therefore, enlist machines to do some of the work**
  - leads to programming languages

- **it's hard to manage the resources of the computer**
- **hard to control sequences of operations**
- **in ancient times, high cost of having machine be idle**
- **therefore, enlist machines to do some of the work**
  - leads to operating systems

Evolution of programming languages

- **1940's: machine level**
  - use binary or equivalent notations for actual numeric values
- **1950's: "assembly language"**
  - names for instructions: ADD instead of 0110101, etc.
  - names for locations: assembler keeps track of where things are in memory; translates this more humane language into machine language
  - this is the level used in the "toy" machine
  - needs total rewrite if moved to a different kind of CPU

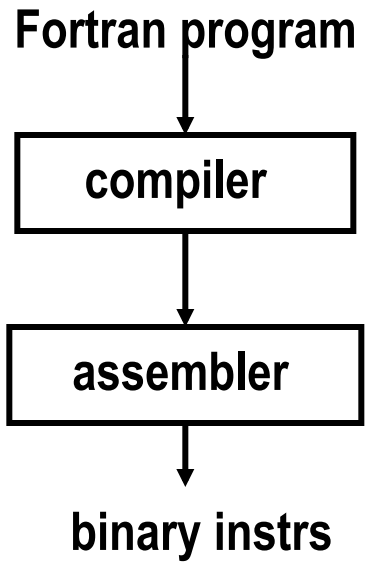
```
loop  get            # read a number
      ifzero  done   # no more input if number is zero
      add     sum    # add in accumulated sum
      store   sum    # store new value back in sum
      goto    loop   # read another number
done  load    sum    # print sum
      print
      stop
sum   0    # sum will be 0 when program starts
```



Evolution of programming languages, 1960's

- **"high level" languages: Fortran, Cobol, Basic**
  - write in a more natural notation, e.g., mathematical formulas
  - a program ("compiler", "translator") converts into assembler
  - potential disadvantage: lower efficiency in use of machine
  - enormous advantages:
    accessible to much wider population of users
    portable: same program can be translated for different machines
    more efficient in programmer time

```
    sum = 0
10  read(5,*) num
    if (num .eq. 0) goto 20
    sum = sum + num
    goto 10
20  write(6,*) sum
    stop
    end
```

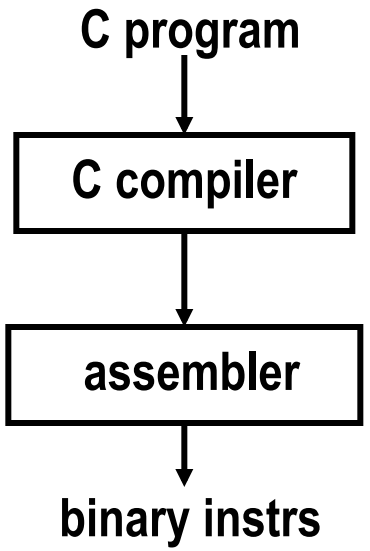


Evolution of programming languages, 1970's

- **"system programming" languages: *C***
  - efficient and expressive enough to take on **any** programming task
    writing assemblers, compilers, operating systems
  - a program ("compiler", "translator") converts into assembler
  - enormous advantages:
    accessible to much wider population of programmers
    portable: same program can be translated for different machines
    faster, cheaper hardware helps make this happen

```
#include <stdio.h>
main() {
  int num, sum = 0;

  while (scanf("%d", &num) != -1 && num != 0)
       sum += num;
  printf("%d\n", sum);
}
```



Evolution of programming languages, 1980's

- **"object-oriented" languages: *C++***
  - better control of structure of really large programs
    better internal checks, organization, safety
  - a program ("compiler", "translator") converts into assembler or *C*
  - enormous advantages:
    portable: same program can be translated for different machines
    faster, cheaper hardware helps make this happen

```
#include <iostream>
main() {
   int num, sum = 0;

   while (cin >> num && num != 0)
        sum += num;
   cout << sum << endl;
}
```

Evolution of programming languages, 1990's

- **"scripting", Web, component-based, ...:**
  Java, Perl, Python, Ruby, Visual Basic, Javascript, ...
  - write big programs by combining components already written
  - often based on "virtual machine": simulated, like fancier toy computer
  - enormous advantages:
    portable: same program can be translated for different machines
    faster, cheaper hardware helps make this happen

```
var sum = 0;  // javascript
var num = prompt("Enter new value, or 0 to end")
while (num != 0) {
       sum = sum + parseInt(num)
       num = prompt("Enter new value, or 0 to end")
}
alert("Sum = " + sum)
```

Programming languages in the 21st century?

- **new general-purpose languages**
  - Go, Rust, Swift, Scala, ...
- **ongoing refinements / evolution of existing languages**
  - C, C++, Fortran, Cobol all have new standards in last few years
- **specialized languages for specific application areas**
  - e.g., R for statistics
- **old languages rarely die**
  - it costs too much to rewrite programs in a new language

Why so many programming languages?

- **every language is a tradeoff among competing pressures**
  - reaction to perceived failings of others; personal taste
- **notation is important**
  - "Language shapes the way we think and determines what we can think about."

    Benjamin Whorf
  - the more natural and close to the problem domain, the easier it is to get the machine to do what you want

- **higher-level languages hide differences between machines and between operating systems**
- **we can define idealized "machines" or capabilities and have a program simulate them -- "virtual machines"**
  - programming languages are another example of Turing equivalence