Ceng 111 – Fall 2015
Week 6b

The world of programming

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.
Program, Programming

Previously on CENG 111!

IMPLEMENTED

```c
int alice = 1;
int bob = 456;
int carol;
main(void)
{
    carol = alice*bob;
    printf("%d", carol);
}
```

PROGRAM
Program, Programming

- Program:
  - “a series of steps to be carried out or goals to be accomplished”

- A recipe for cooking a certain dish is also a program (but not a computer program).
The Translation/Loading/Execution Process
Assembly Language (continued)

- Source program
  - An assembly language program

- Object program
  - A machine language program

- Assembler
  - Translates a source program into a corresponding object program
Assembly Language (continued)

Advantages of writing in assembly language rather than machine language

- Use of symbolic operation codes rather than numeric (binary) ones
- Use of symbolic memory addresses rather than numeric (binary) ones
- Pseudo-operations that provide useful user-oriented services such as data generation
BEGIN  --This must be the first line of the program.
  --Assembly language instructions like those in Figure 6.5.
HALT  --This instruction terminates execution of the program
  --Data generation pseudo-ops such as
  --.DATA are placed here, after the HALT.
.END   --This must be the last line of the program.

Structure of a Typical Assembly Language Program
Examples of Assembly Language Code

- Arithmetic expression

A = B + C − 7

(Assume that B and C have already been assigned values)
Examples of Assembly Language Code (continued)

Assembly language translation

LOAD B --Put the value B into register R.
ADD C --R now holds the sum (B + C).
SUBTRACT SEVEN --R now holds the expression (B + C - 7).
STORE A --Store the result into A.

: --These data should be placed after the HALT.

A: .DATA 0
B: .DATA 0
C: .DATA 0
SEVEN: .DATA 7 --The constant 7.
Examples of Assembly Language Code (continued)

Problem

- Read in a sequence of non-negative numbers, one number at a time, and compute their sum

- When you encounter a negative number, print out the sum of the non-negative values and stop
<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set the value of Sum to 0</td>
</tr>
<tr>
<td>2</td>
<td>Input the first number $N$</td>
</tr>
<tr>
<td>3</td>
<td>While $N$ is not negative do</td>
</tr>
<tr>
<td>4</td>
<td>Add the value of $N$ to Sum</td>
</tr>
<tr>
<td>5</td>
<td>Input the next data value $N$</td>
</tr>
<tr>
<td>6</td>
<td>End of the loop</td>
</tr>
<tr>
<td>7</td>
<td>Print out Sum</td>
</tr>
<tr>
<td>8</td>
<td>Stop</td>
</tr>
</tbody>
</table>

Algorithm to Compute the Sum of Numbers
Assembly Language Program to Compute the Sum of Nonnegative Numbers
Translation and Loading

- Before a source program can be run, an assembler and a loader must be invoked

- **Assembler**
  - Translates a symbolic assembly language program into machine language

- **Loader**
  - Reads instructions from the object file and stores them into memory for execution
Translation and Loading (continued)

Assembler tasks

- Convert symbolic op codes to binary
- Convert symbolic addresses to binary
- Perform assembler services requested by the pseudo-ops
- Put translated instructions into a file for future use
main:
pushq  %rbp
movq   %rsp, %rbp
movl   alice(%rip), %edx
movl   bob(%rip), %eax
imull  %edx, %eax
movl   %eax, carol(%rip)
movl   $0, %eax
leave
ret

alice:
.long  123

bob:
.long  456
There is a limit to how high a language can get.

Why can’t we write programs in our spoken language?
Programming Language Paradigms

Classification / Categorization of programming languages.

- Imperative Paradigm
- Functional Paradigm
- Logical-declarative Paradigm
- Object-oriented Paradigm
- Concurrent Paradigm
- Event-driven Paradigm
Imperative Paradigm

Statement_1
Statement_2
Statement_3
Statement_4
Statement_5

From C:

```c
int a = 2;
int b = a * 2;
int c;

c = -b - sqrt(b*b - 4*a*c) / (2*a);
```
Functional Paradigm

- Data environment is restricted.
- Functions receive their inputs and return their results to the data environment.
- Programmer’s task:
  - decompose the problem into a set of functions such that the composition of these functions produce the desired result.
Functional Paradigm

Imperative Version of Fibonacci Numbers

```
# Fibonacci numbers, imperative style
N=10

first = 0  # seed value fibonacci(0)
second = 1  # seed value fibonacci(1)

fib_number = first + second    # calculate fibonacci
for position in range(N-2):   # iterate N-2 times
    first = second            # update the value
    second = fib_number
    fib_number = first + second # update the result

print fib_number
```

Functional Version of Fibonacci Numbers

```
# Fibonacci numbers, functional style
def fibonacci(N):  # Fibonacci number N (for N > 0)
    if N <= 1: return N  # base cases
    else: return fibonacci(N-1) + fibonacci(N-2)

print fibonacci(10)
```
Logical-declarative Paradigm

- The data and the relations are states as rules, or facts.
- The problem is solved by writing new rules/facts.

From Prolog:

```
mother(matilda,ruth).
mother(trudi,paggy).
mother(eve,alice).
mother(zoe,sue).
mother(eve,trudi).
mother(matilda,eve).
mother(eve,carol).
grandma(X,Y) :- mother(X,Z), mother(Z,Y).
```
Object Oriented Paradigm

CAR Class

Data
- Wheels: Wheel₁, Wheel₂, ..., Wheelₙ
- Gear: Manual, Automatic, Triptronic
- Steering: Hydraulic, Electrical, Mechanical
- Doors: Door₁, Door₂, ..., Doorₘ
- Size: Width, Height
- Fuel: Gasoline, Diesel, LPG
- Engine: K Valves, L Cylinders

Actions
- Steer
- Brake
- Speed up

CAR Objects - Instances of CAR Class

- Car Object 1
- Car Object 2
- Car Object 3
- Car Object 4
- Car Object 5
- Car Object 6
Object Oriented Paradigm

- Problem is decomposed into objects which hold data and the corresponding functions on the data.

- Objects can be defined using other objects as a basis; the new object inherits from the basis objects.

```cpp
class Item {
    string Name;
    float Price;
    string Location;
    ...;
};

class Book : Item {
    string Author;
    string Publisher;
    ...
};

class MusicCD : Item {
    string Artist;
    string Distributor;
    ...
};
```
Today

- Continue on the world of programming

**Reminder:**
- Tentative midterm date: 9 December at 17:40.
Concurrent Paradigm

- Programming using multiple CPUs concurrently.
- The task is to assign the overall flow & data to individual CPUs.
- With the bottleneck in CPU power, this paradigm is going to be the trend in the future.
Event-Driven Paradigm

- A program is composed of events and what to do in case of events.
- The task is to decompose a problem into a set of events and the corresponding functionalities that will be executed in case of events.
- Suitable for Graphical User Interface design.
The hyperspace of languages

Figure 1.4: The hyperspace of programming (only 3 axes displayed)
Zoo of Programming Languages

- Around 700 programming languages!

- But, why do we not have a programming language that serves all paradigms/purposes/requirements?
Choosing a PL

- Ex: Moving soil with a shovel and a grader
Factors that affect choosing a PL

- Domain & Technical Nature of the Problem

  a) Finding the pixels of an image with RGB value of [202,130,180] with a tolerance of 5.4% in intensity.
  b) A proof system for planar geometry problems.
  c) A computer game platform which will be used as a whole or in parts and may get extended even rewritten by programmers at various levels.
  d) Payroll printing.
Factors that affect choosing a PL

- Personal taste and preference
- Circumstance-imposed constraints
  - e.g., time limit.
- Current trend
Current trend

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
How are languages implemented

```
int alice = 1;
int bob = 456;
int carol;
main(void)
{
    carol = alice*bob;
    printf("%d", carol);
}
```

- **Compiler**
- **Object Code**
- **Linker**
- **Executable Code**

**Compilation Process**:
1. **Source Code**
2. **Compiler** transforms the source code into **Object Code**.
3. **Linker** combines the object code with other object codes and libraries to produce the **Executable Code**.
How are languages implemented

```c
int alice = 123;
int bob = 456;
int carol;
main(void)
{
    carol = alice*bob;
    printf("%d", carol);
}
```
How are languages implemented

INTERPRETIVE APPROACH

USER TYPES
int alice = 123;
INTERPRETER REPLIES
OK. alice is 123.

USER TYPES
int bob = 456;
INTERPRETER REPLIES
OK. bob is 456.

USER TYPES
print bob * alice;
INTERPRETER REPLIES
56088
Now

- Testing
- Errors and debugging
- Python
How is a program written?

- Modular & Functional Breakdown
- For example:
  - User interface module
  - Database module
  - Control module
Testing

- Top-down Testing

Top-Down Integration Testing

http://sce.uhcl.edu/whiteta/sdp/subSystemIntegrationTesting.html
Testing

- Bottom-up Testing

http://sce.uhcl.edu/whiteta/sdp/subSystemIntegrationTesting.html
Testing

- Black-box Testing

Input → The System → Output

Requirements
Testing

- White-box Testing

Requirements

The System

Input

Output

2015 S. Kalkan & G. Ucoluk & A. Cosar - CEng III
The world of programming

- Types of errors
- Meet Python

Data

- Basic Data
- Structured Data
Bugs, Errors

- **Syntax Errors**
  
  \[ \text{Area} = 3.1415 \times R \times R \]
  
- **Run-time Errors**

```python
>>> def SqrtDelta(a, b, c):
    return sqrt(b*b - 4*a*c)

>>> print SqrtDelta(1,3,1)
2.2360679774997898
>>> print SqrtDelta(1,1,1)
ValueError: math domain error
```
Bugs, Errors

- Logical Errors

\[ \text{root}_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]

>>> root1 = (-b + sqrt(b*b - 4*a*c)) / 2*a

- Design Errors

\[ x^3 + ax^2 + bx + c = 0 \]

\[ \text{root}_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]
An interpretive/scripting PL that:
- Longs for code readability
- Ease of use, clear syntax
- Wide range of applications, libraries, tools

Multiple Paradigms:
- Functional
- Imperative
- Object-oriented
- Started at the end of 1980s.
- V2.0 was released in 2000
  - With a big change in development perspective: Community-based
  - Major changes in the facilities.
- V3.0 was released in 2008
  - Backward-incompatible
  - Some of its features are put into v2.6 and v2.7.
Where does the name come from?

One goal of Python: “fun to use”

- The origin of the name is the comedy group “Monty Python”
- This is reflected in sample codes that are written in Python by the original developers.
skalkan@divan:~$ python
Python 2.5.2 (r252:60911, Jan 24 2010, 17:44:40)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>