PRINCIPLES OF PROGRAMMING LANGUAGES

Names, Bindings and Scope

Prafullata Kiran Auradkar
Computer Science and Engineering

PRINCIPLES OF PROGRAMMING LANGUAGES

Names, Bindings and Scope

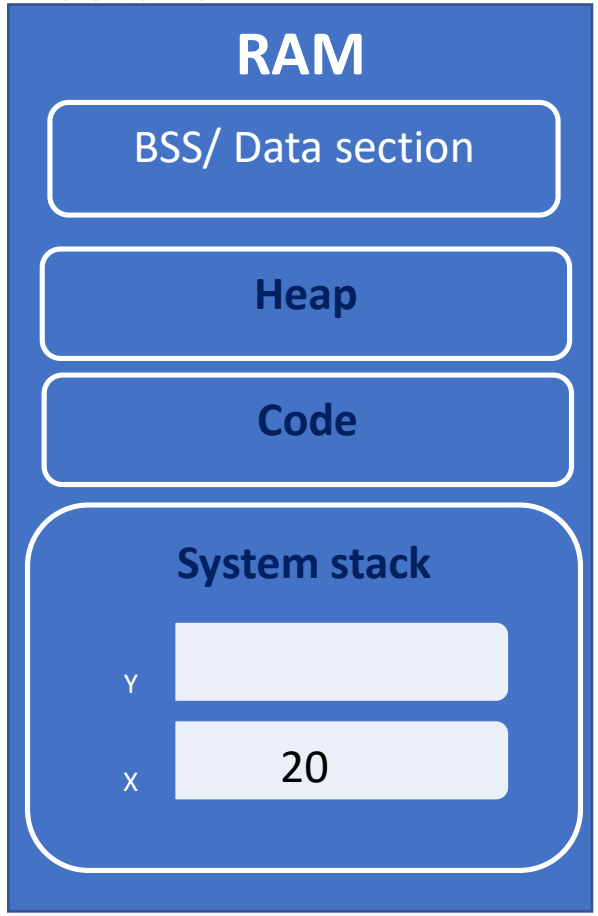
Prafullata Kiran Auradkar
Computer Science and Engineering

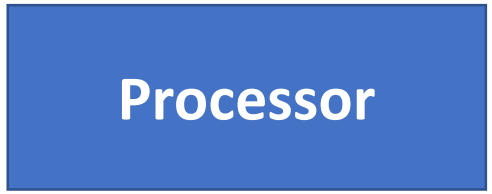
PRINCIPLES OF PROGRAMMING LANGUAGES

Names, Bindings and Scope

- Imperative languages are abstractions of von Neumann architecture
  - Memory
  - Processor

```
#include<stdio.h>
int main()
{
int x=20, y=x;
printf("%d\n%d\n",x,y);
return 0;
}
```





Variables

- Variables characterized by attributes
  - Name
  - Type
  - Scope
  - Lifetime
  - Address
  - Value

Names: Identifier naming

- Design issues for names:
  - Maximum length?
  - Are connector characters allowed?
  - Are names case sensitive? – readability ?
  - Are special words reserved words or keywords?
    - Context sensitive
    - Contextual keywords…

Variable attributes

- Name - not all variables have them
  - Dynamic variables (C) – through malloc…
- Address - the memory address with which variable is associated
  - A variable may have different addresses at different times during execution
  - A variable may have different addresses at different places in a program
  - If two variable names can be used to access the same memory location, they are called aliases
    - Aliases are created via pointers, reference variables, C and C++ unions

Variable attributes

- A variable may have different addresses at different times during execution

```
int fact(int x)
{
if(x==1)
return 1;
else
return (x *
   fact(x-1));
}
```

```
int main()
{
int f=fact(3);
… }
```

stack
X=1, return 1
X=2, 2 * (return..)
X=3, 3 * (return..)
Return to main

Variable attributes

- A variable may have different addresses at different places in a program

```
int f(int x)
{
int m = x;
…
{
int m=100;
…
}

}
```

Variable attributes

- If two variable names can be used to access the same memory location, they are called aliases
  - Aliases are created via pointers, reference variables, C and C++ unions
  - C and Python examples for pointers and reference types
  - (Union example in next Unit…)

Variable attributes

- *Type* - determines the range of values of variables and the set of operations that are defined for values of that type; in the case of floating point, type also determines the precision
- *Value* - the contents of the location with which the variable is associated
  - *Abstract memory cell* - the physical cell or collection of cells associated with a variable

Binding

- A **binding** is an association, such as between an attribute and an entity, or between an operation and a symbol
- **Binding time** is the time at which a binding takes place.

Possible Binding Times

- Language design time -- bind operator symbols to operations
- Language implementation time-- bind floating point type to a representation
- Compile time -- bind a variable to a type in C or Java
- Load time -- bind a FORTRAN 77 variable to a memory cell (or a C `static` variable)
- Runtime -- bind a non-static local variable to a memory cell

Possible Binding Times

- A binding is **static** if it first occurs before run time and remains unchanged throughout program execution.
- A binding is **dynamic** if it first occurs during execution or can change during execution of the program

Type Binding

- How is a type specified?
- When does the binding take place?
- If **static**, the type may be specified by either an explicit or an implicit declaration
  - Examples
    - C (Explicit)
    - Fortran (Implicit)

```
int f(int x)
{
int m = x;
…
{
int m=100;
…
} }
```

Type Binding - Dynamic

- Dynamic Type Binding (JavaScript, PHP, Python)
- Specified through an assignment statement e.g., JavaScript

```
list = [2, 4.33, 6, 8];
list = 17.3;
```

  - Advantage: flexibility (generic program units)
  - Disadvantages:
    - High cost (dynamic type checking and interpretation)
    - Type error detection by the compiler is difficult

Type Inferencing

- Examples – Haskell Functions

Value Binding - Initialization

- Examples – C, Java, C++

Do It Yourself

- Auto initialization in C for,
  - Array variable
  - Global variable
    - Only constants allowed or can it be dynamic?
  - Static variable in global & local scopes

THANK YOU

Prafullata Kiran Auradkar
Computer Science and Engineering

prafullatak@pes.edu