Lecture 7

Overview of Types and Data Representation

Variables

- A program variable is an abstraction of a computer memory cell or collection of cells.
- A variable is a name written in high level language which makes the program more readable.
- A variable can represent memory locations to hold specific set of values characterized by its type.

Variables, declaration statements and memory addresses

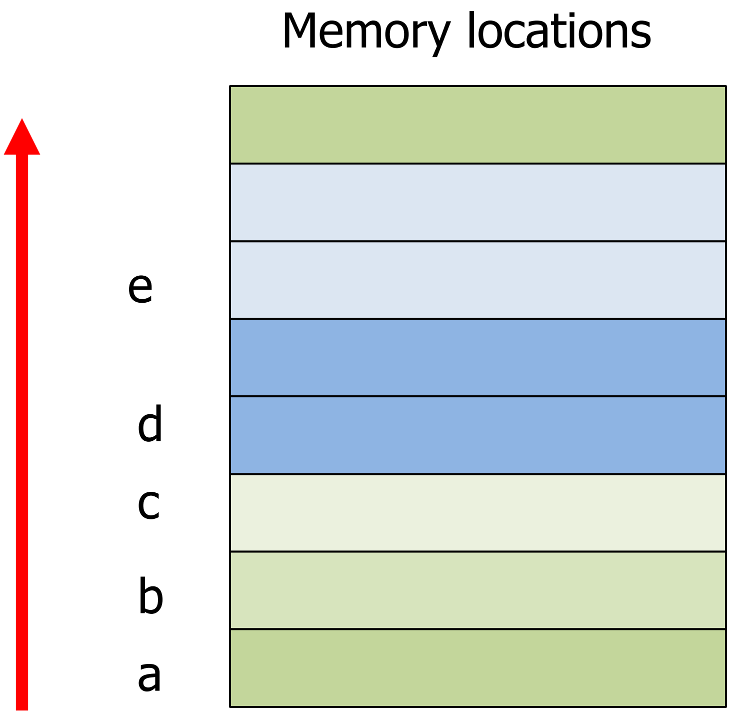
- Variable names and types are written in declaration statements by the user in the program.

Program code

```
int a, b, c;
float d,e;
```

Grammar

```
<DeclStmt>→<Type> <list>
<Type>→ INT | FLOAT| CHAR
<list>→ <list> COMMA ID | ID
```



Variables and its attributes

- Name
- Type
- Address
- Value
- Lifetime
- scope

Types- Motivation

A sample code that adds two numeric values

```
X=5;
Y=53.7523;
Z=X+Y;
```

A sample code that concatenates two strings and prints the string

```
strcpy(name,"CS");
strcat(name,"IS");
printf("%s\n",name);
```

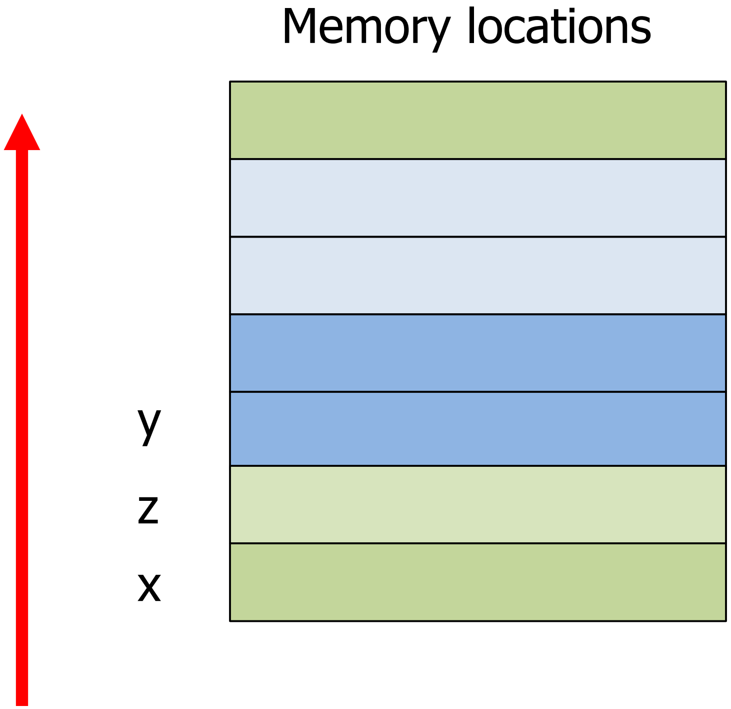
Addition of two different types of data

Operation on same type of data

Different types of data

- Adding different types of data and storing in a variable leads to precision loss.

```
int x, z;
float y;
x=5;
y=53.7523;
z=x+y;
```



Combining two types of data with an operator to get resultant meaningless value

- i=23+”programming”
- strcat(name,20.5)

•The validity check on the above two operations can be enforced by associating a type with names and corresponding values.

•The addition of an integer with a string can be reported as an error by verifying and enforcing type consistency.

Types- motivation

- Types classify “things” in any domain
- Associating type with a name of a variable defines
  - the possible values that the variable can take, and
  - the possible operations that can be performed on the values

  Example:

  int x, y, sum;   //with operations +,-,*,/ ,%

boolean flag;   //with logical operations and, or, not etc

Advantages of associating a type with a variable name

- More readability
- Protection through type consistency checks at compile time
- Size of data object can be inferred from its type
- Unsafe/Invalid operations are avoided (protection)

Example

- Represent of a day in a year by an integer
- If January 31 is represented as 31
  - March 15 is represented as 74
  - December 31 as 365
- Let variable d represent the day as number n
- Operations
  - tomorrow(d) = n+1
  - yesterday(d) = n-1
- Since the type of the day is inappropriately chosen as integer, and * is a valid operation on integers, d1 * d2 will have a value with no meaning

Types- basic definitions

- Data types are the sets of values along with a set of associated operations
- Typing (i.e. type checking) is membership
- Example:

  int x;

  ⇒ x can take any value from the set

  int={minint, .., -1,0,1,..,maxint}

  where minint and maxint are machine dependent values

Type

- A type of a ‘symbol’ is defined as the set of values such that
  - There exists a common collection of operations on these values
  - Values share a common representation
- Example:
  - x=-5; y=4; z=x+y

X and y values being of int type produce the sum as -1, which is int type

Primitive Data objects

- Directly manipulated by the underlying machine
- Integers and other primitive values are the first class citizens
- Operations on basic values are built into the languages
- Programmer defined data objects are constructed from simpler types