Variables, memory allocation (Variables and data representation)

CSC 206
FOUNDATION OF SEQUENTIAL PROGRAMS

DISCLAIMER

The contents of this document are intended for leaning purposes at the undergraduate level. The materials are from different sources including the internet and the contributors do not in any way claim authorship or ownership of them.

Variables

- A symbol or name that stands for a value.
- A **variable** is a value that can change.
- Variables provide **temporary storage** for information that will be needed during the lifespan of the computer program (or application).

Variables

Example:

$$z = x + y$$

- This is an example of programming expression.
- x, y and z are variables.
- Variables can represent numeric values, characters, character strings, or memory addresses.

Variables

- Variables store everything in your program.
- The purpose of any useful program is to modify variables.
- In a program every, variable has:
  - Name (Identifier)
  - Data Type
  - Size
  - Value

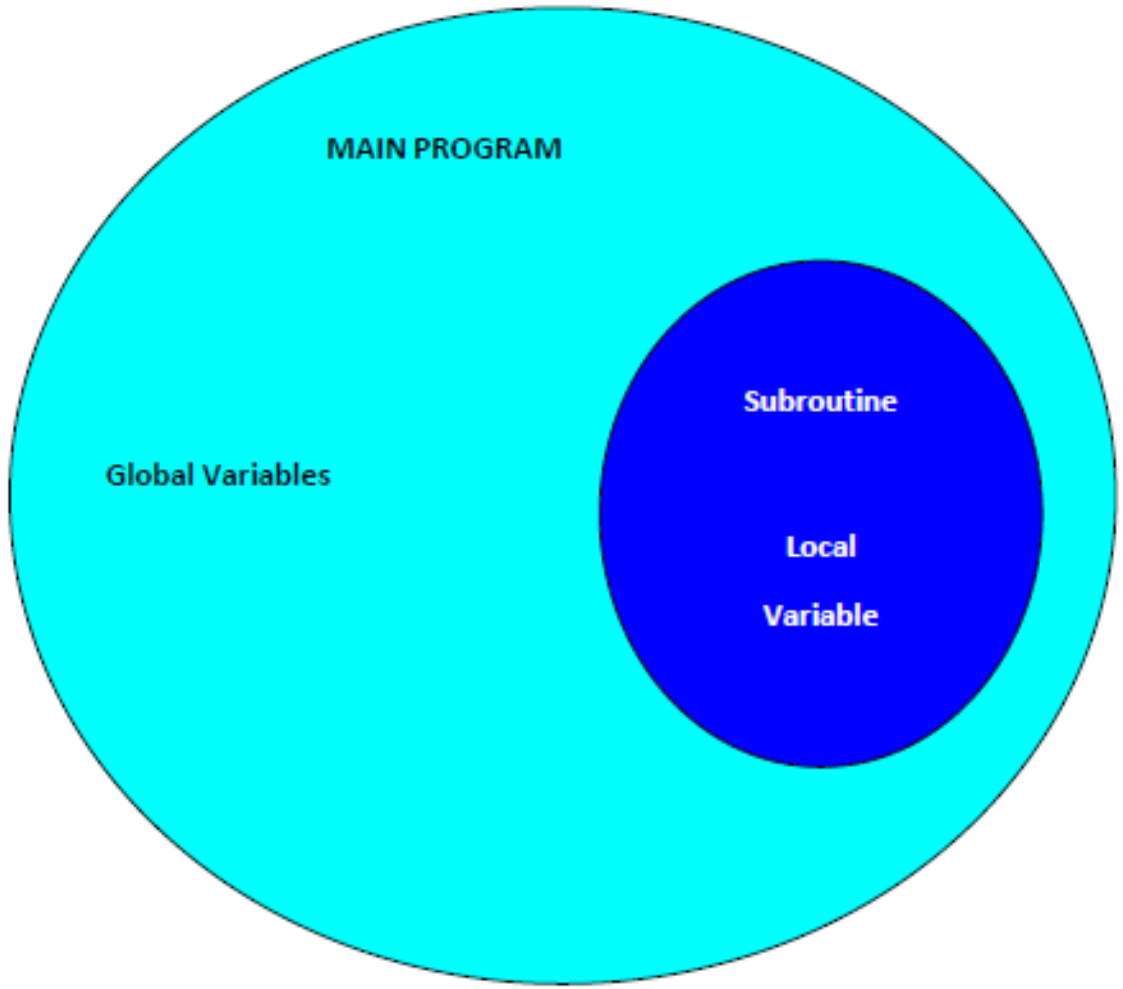
Types of Variable

- There are two types of variables:
  - Local variable
  - Global variable

Types of Variable

- **Local variables** are those that are in scope within a specific part of the program (function, procedure, method, or subroutine, depending on the programming language employed).
- **Global variables** are those that are in scope for the duration of the programs execution. They can be accessed by any part of the program, and are read-write for all statements that access them.

Types of Variable



Rules in Naming a Variable

- There a certain rules in naming variables (identifier).
- They are:
  - It should use only alphabets, number and underscore (_)
  - It should not begin with a number and must have at least one alphabet.
  - It should not be a reserved word.
- Examples of reserved word:

main	long	if	do	continue
short	else	return	const	int
double	break	void	while	char

Rules in Naming a Variable

- Following are some valid identifiers:

x number1 a123 number_2 _xyz

- Following are some invalid identifiers:

print main	space not allowed
123number	name should not begin with number
main	reserved word

Exercise

Explain these mathematical problems by using variables.

1. Area of square
2. Area of triangle
3. Area of circle
4. Average speed

Constants

- The opposite of a variable is a constant.
- A constant is a value that **never changes**.
- Because of their inflexibility, constants are used less often than variables in programming.
- A constant can be :
  - a number, like 25 or 3.6
  - a character, like *a* or *$*
  - a character string, like "this is a string"

Constants

- Example:
  - Calculate area of circle

$$area = 3.142 * r * r$$

- Variable:
  - area, r
- Constant
  - 3.142

Data Types

- Data type is classification of a particular type of information.
- Data types are essential to any computer programming language.
- Without them, it becomes very difficult to maintain information within a computer program.
- Different data types have different sizes in memory depending on the machine and compilers.

Data Types

- Integer
- Floating-point
- Character
- String
- Boolean

Integer

- A whole number, a number that has no fractional part.
- The following are integers:

0 1 -125 144457

- In contrast, the following are not integers:

5.34 -1.0 1.3E4 "string"

- There are often different sizes of integers available; for example, PCs support short integers, which are 2 bytes, and long integers, which are 4 bytes.

Floating-point

- A number with a decimal point.
- The following are floating-point numbers:

3.0 -111.5

- Floating-point representations are slower and less accurate than fixed-point representations, but they can handle a larger range of numbers.
- It require more space.

Character

- In graphics-based applications, the term **character** is generally reserved for letters, numbers, and punctuation..
- It only can represent single character.
- The following are characters:

a A @ $ 2

String

- String is a sequence of character.
- Example:
  - Kuala Lumpur
  - John White
  - Computer

Boolean

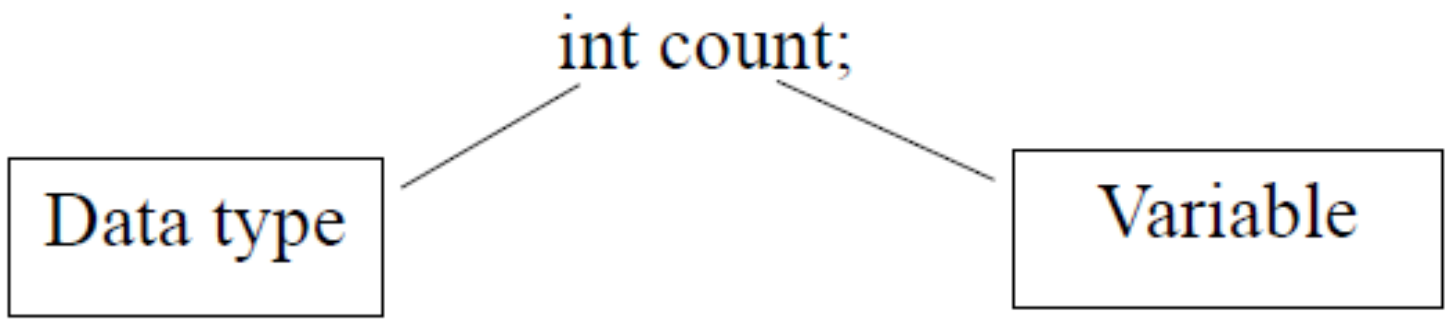
- This data type only have two possible values:
  - True
  - False
- Use this data type for simple flags that track true/false conditions.

Declaration

- In programming languages all the variables that a program is going to use must be **declared** prior to use.
- Declaration of a variable serves two purposes:
  - It associates a **type** and an identifier (or name) with the variable. The type allows the compiler to interpret statements correctly.
  - It allows the compiler to decide how much storage space to allocate for storage of the value associated with the identifier and to assign an address for each variable which can be used in code generation.

Declaration

- A typical set of variable declarations that might appear at the beginning of a program could be as follows:



Declaration

- Often in programming numerical constants are used, e.g. the value of π (pi = 3.14).
- It is well worthwhile to associate meaningful names with constants.
- These names can be associated with the appropriate numerical value in a **constant declaration**.
- The names given to constants must conform to the rules for the formation of identifiers as defined above.

Declaration

- The following constant declaration:

```
const int days_in_year = 365;
```

- The general form of a constant declaration is:

const *type constant-identifier* = *value* ;

Assignment

Are these identifiers valid or not? Explain.

1. a1
2. Number3
3. 123abc
4. a_123
5. no 2

Identify the data type:

1. 312
2. 2.153
3. a1
4. a
5. True
6. hello

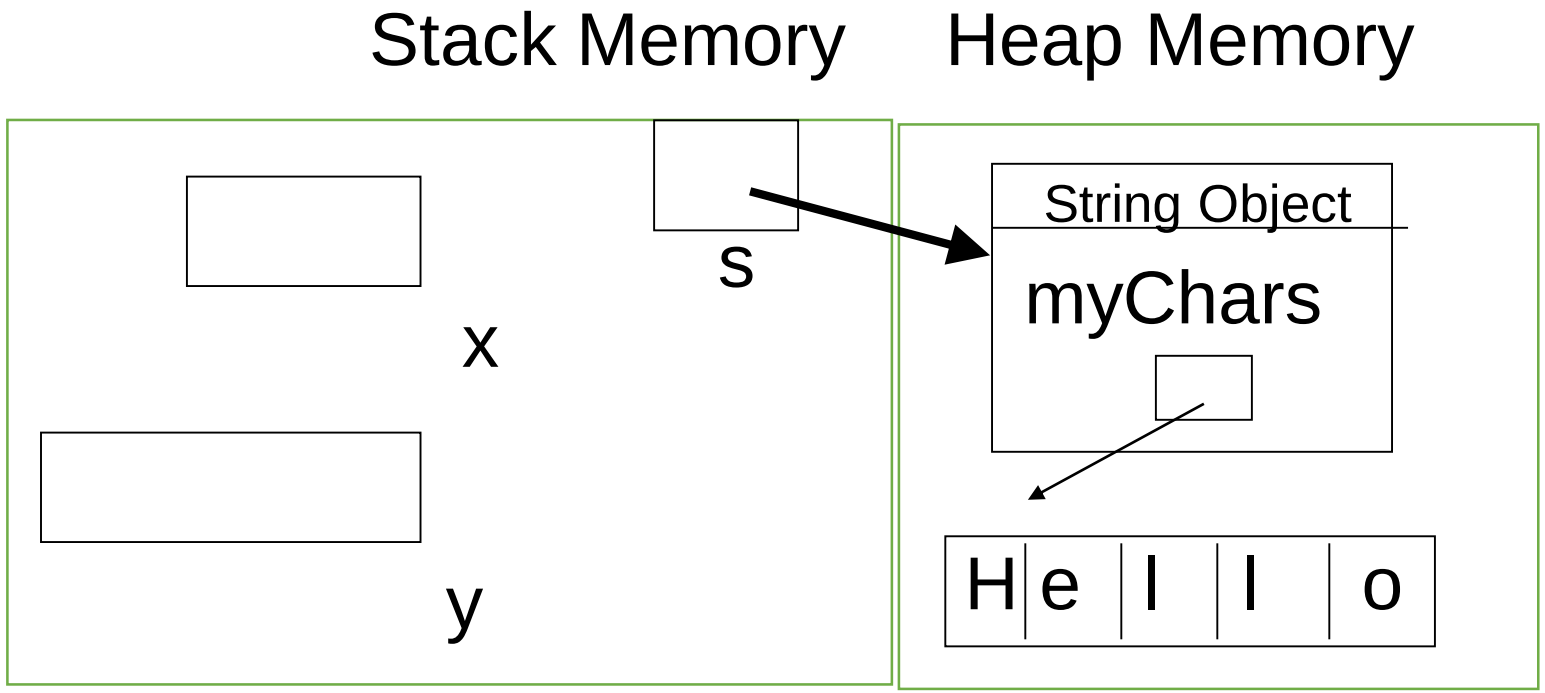
Memory

Your programs /software has two main kinds of memory to work with, which are:

- Stack memory

- Heap memory

The Picture




```
void toyCodeForMemory(int x)
{    int y = 10;
     x += y;
     String s = new String("Hello");
     System.out.println(x + " " + y + s);
}
```

Stack memory

- Local variables for functions, whose size can be determined at call time.
- Information saved at function call and restored at function return:
  - Values of called arguments
  - Register values:
    - Return address (value of PC)
    - Frame pointer (value of FP)
    - Other registers

Heap memory

- Structures whose size varies dynamically (e.g. variable length arrays or strings).
- Structures that are allocated dynamically (e.g. records in a linked list).
- Structures created by a function call that must survive after the call returns.

Issues:

- Allocation and free space management
- Deallocation / garbage collection

Memory Allocation

Memory allocation in computer science, is the act of allocating memory to a program for its usage, typically for storing variables, code or data. Memory allocation is a widely discussed topic in the field of operating systems, as computers have limited memory, and many programs need memory to run.

Memory Allocation

- There are essentially two types of memory allocation
  - Static – Done by the compiler automatically (implicitly).
    - Global variables or objects -- memory is allocated at the start of the program, and freed when program exits; alive throughout program execution
      - Can be access anywhere in the program.
    - Local variables (inside a subroutine) – memory is allocated when the subroutine starts and freed when the subroutine returns.
      - A local variable cannot be accessed from another subroutine.
    - Allocation and free are done implicitly.
    - No need to explicitly manage memory, but has limitations!
      - When you use static allocation, the array size must be fixed.

Memory Allocation

- I think it would be nice to have an array whose size can be adjusted depending on one needs.
  - Dynamic memory allocation deals with this situation.

- Dynamic – Done explicitly by programmer.
  - Programmer explicitly requests the system to allocate memory and return starting address of memory allocated. This address can be used by the programmer to access the allocated memory.
  - When done using memory, it must be **explicitly** freed.

Explicitly allocating memory in C++: The ‘new’ Operator

- Used to dynamically allocate memory
- Can be used to allocate a single variable/object or an array of variables/objects
- The new operator returns pointer to the type allocated
- Examples:
  - char *my_char_ptr = new char;
  - int *my_int_array =new int[20];
  - Mixed *m1 = new Mixed(1,1,2);

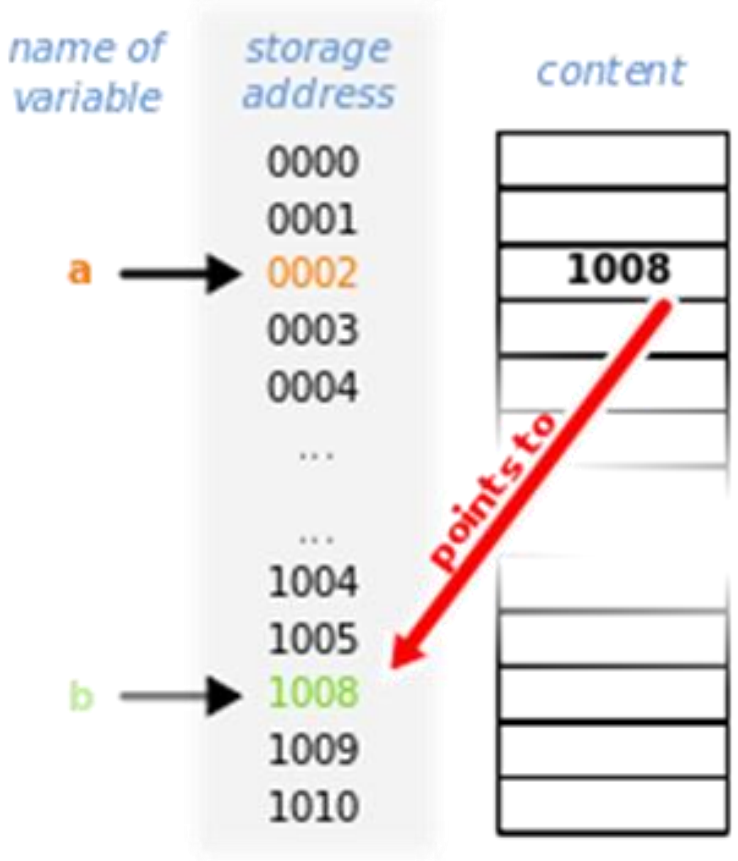
  - Before the assignment, the pointer may or may not point to a legitimate memory
  - After the assignment, the pointer points to a legitimate memory.

What is a pointer?

In computer science, a pointer is a variable whose value is a location in the computer's memory. For example If Mr Brown stands in a room and points to his friend Sele, then Mr Brown is a pointer whose value is Sele’s location. A programmer must dereference the pointer to retrieve the object it points to. Pointers do not take up much memory (RAM).

Copying a pointer to a large object is faster than copying the large object itself because only the location needs to be copied, instead of the whole object.

Description of a pointer



Explicitly freeing memory in C++: the ‘delete’ Operator

- The delete operator is used to free memory allocated with new operator
- The delete operator should be called on a pointer to dynamically allocated memory when it is no longer needed
- How can one delete a single variable/object in an array
  - delete PointerName;
  - delete [] ArrayName;
- After delete is called in a memory region, that region should no longer be accessed by the program
- The general convention is to set pointer of deleted memory to NULL
  - Any new must have a corresponding delete --- if not, the program has memory leak.
  - New and delete may not be in the same subroutine/function.

Explicit memory allocation/free in C (Works also in C++)

- The malloc and free routines
  - malloc is similar to new except that it specifies the exact memory size
    - Return a (void *) -- needs to convert to the right pointer type
  - free is equivalent to delete (only one form for both single item and many items).

Why use dynamic memory allocation?

- Allows data (especially arrays) to take on variable sizes (e.g. ask the user how many numbers to store, then generate an array of integers exactly that size).
- Allows locally created variables to live past end of subroutine.
- Allows us to create many structures used in Data Structures and Algorithms