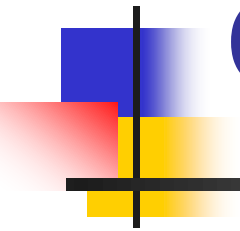
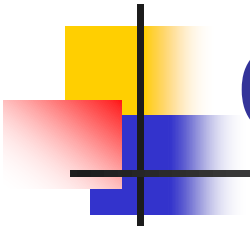


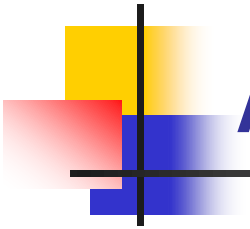
Chapter 1

Review of C++ concepts



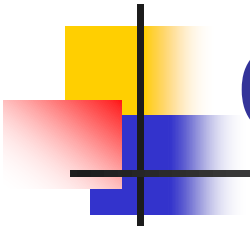
Contents

- Arrays
- Structures
- Functions
- Pointers



Arrays

- An array
  - a single name for a collection of data values
  - all of the same data type
  - subscript notation to identify one of the values
- A carryover from earlier programming languages
- More than a primitive type, less than an object
  - like objects when used as method parameters and return types
  - do not have or use inheritance
- Accessing each of the values in an array
  - Usually a `for` loop



Cont..

Accessing arrays is similar:

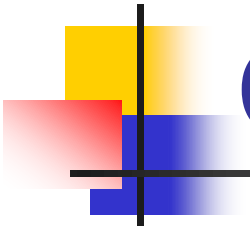
```
const int ARRAY_CAPACITY = 10; // prevents reassignment
int array[ARRAY_CAPACITY];

array[0] = 1;
for ( int i = 1; i < ARRAY_CAPACITY; ++i ) {
    array[i] = 2*array[i - 1] + 1;
}
```

Recall that arrays go from **`0`** to **`ARRAY_CAPACITY - 1`**

Definition:

The *capacity* of an array is the entries it can hold

The *size* of an array is the number of useful entries



Creating Arrays

- General syntax for declaring an array:

```
Base_Type[] Array_Name = new Base_Type[Length];
```

- Examples:
  80-element array with base type `char`:

```
char[] symbol = new char[80];
```

  100-element array of `doubles`:

```
double[] reading = new double[100];
```

  70-element array of `Species`:

```
Species[] specimen = new Species[70];
```

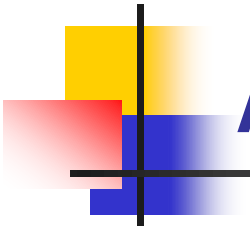
Three Ways to Use [ ] (Brackets) with an Array Name

1. Declaring an array: `int[] pressure`
   - creates a name of type "`int` array"
   - types `int` and `int[]` are different
     - `int[]`: type of the array
     - `int` : type of the individual values

2. To create a new array, e.g. `pressure = new int[100];`

3. To refer to a specific element in the array
   - also called *an indexed variable*, e.g.

```
pressure[3] = keyboard.nextInt();
System.out.println("You entered" + pressure[3]);
```



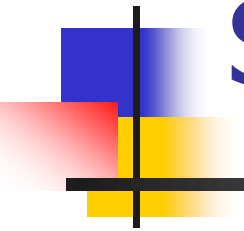
Array Length

- Specified by the number in brackets when created with `new`
  - *maximum* number of elements the array can hold
  - storage is allocated whether or not the elements are assigned values

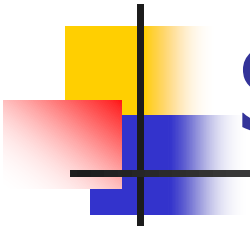
- the attribute `length`,

```
Species[] entry = new Species[20];
System.out.println(entry.length);
```

- The `length` attribute is established in the declaration and cannot be changed unless the array is redeclared



Structure



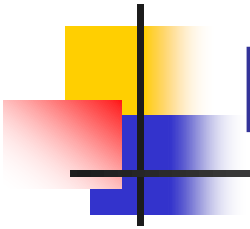
Structure Basics

- C++ structures allow variables to be grouped to form a new data type.
- The data elements in a structure are arranged in a manner that is similar to the way database programs arrange data.
- C++ allows you to create a record by grouping the variables and arrays necessary for the fields into a single structure.

Declaring a Structure

```
struct inventory_item
    {
      apstring item_ID;
      apstring description;
      int quantity_on_hand;
      int reorder_point;
      double cost;
      double retail_price;
    };
```

Declaring a Structure



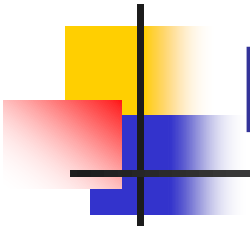
- Once you have declared a structure, you must declare an identifier, called a structure variable, that is of the structure's type.
- The statement below creates a structure variable of a structure named `todays_special` that is of type `inventory_item`.

```
inventory_item todays_special;
```

Accessing Members of a Structure

- To access a member of the structure, use the name of the variable, the dot operator, then the name of the member you need to access.

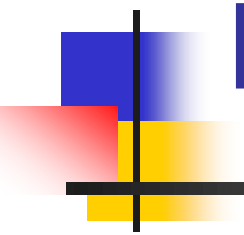
```
todays_special.cost=47.80;
```



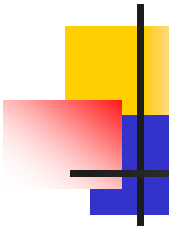
Nested Structures

- A structure can include enumerated data types and even other structures as members.
- Accessing the nested structure requires that two periods be used.
- example:

```
current_donor.bp.diastolic = 74;
```



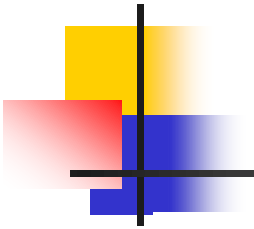
Functions



FUNCTIONS

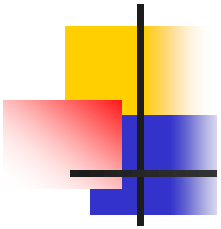
```
Void main()
{
Statement 1;
Statement 2;
Statement 3;
.
.
.
Statement n;
}
```

```
Void main()
{
Statement 1;
Statement 2;
Sum1();
Statement 3;
Statement 4;
Sum2();
Statement 5;
Statement 6;
}
```



Advantages

- Support for modular programming
- Reduction in program size.
- Code duplication is avoided.
- Code reusability is provided.
- Functions can be called repetitively.
- A set of functions can be used to form libraries.



Types

1.Built in functions :-
are part of compiler package.
Part of standard library made available by compiler.
Can be used in any program by including respective header file.

2. User defined functions:-
Created by user or programmer.
Created as per requirement of the program.

Parts of a function

```
main function
{
function prototype declaration
function call
-------;
}

function declaratory/definition
{
------;
return statement
}
```

Function prototype

A function prototype is a declaration of a function that tells the program about the type of value returned by the function, name of function, number and type of arguments.

Syntax: Return_type function_name (parameter list/argument);

```
int    add(int,int);
void   add(void);
int    add(float,int);
```

4 parts

i. Return type
ii. Function name
iii. Argument list
iv. Terminating semicolon

Variable declaration

```
Data_type variable_name ;
int x=5;
float marks;
int price;
```

Function call

A function must be called by its name followed by argument list enclosed in semicolon.

Syntax: function_name (parameter list/argument);

add(x,y);
add(40,60);
add(void); or add();

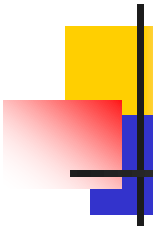
Note: data type not to be mentioned.

Suppose int add(int,int); //prototype

Now to this function add(x,y); //function call

or

add(40,60);



Parts of a function

main function

{

function prototype declaration

function call

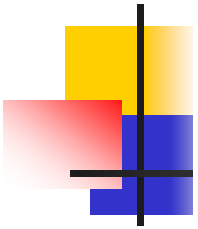
}

function declaratory/definition

{

return statement

}



Function categories

i) Function with no return value and no argument.

```
void add(void);
```

ii) Function with arguments passed and no return value.

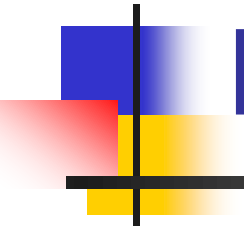
```
void add(int,int);
```

iii) Function with no arguments but returns a value.

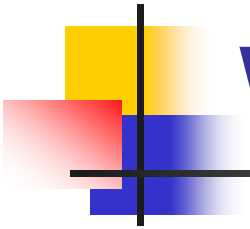
```
int add(void);
```

iv) Function with arguments and returns a value.

```
int add(int,int);
```

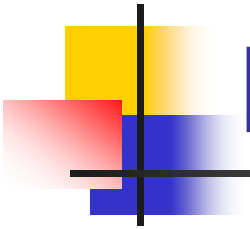


Pointer



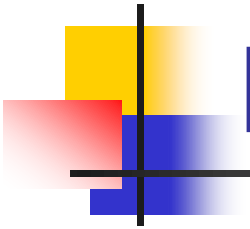
What is a Pointer?

- A pointer is a variable or constant that holds a memory address.
- Think of a pointer as a variable or constant that points to another variable or data structure.
- Pointers can be used to work with multiple variables or advanced arrays.



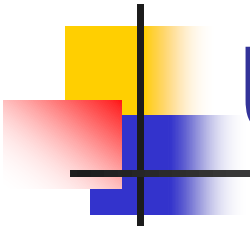
Declaring Pointers

- Working with pointers requires the use of two new operators: the dereferencing operator (*) and the address-of operator (&).
- Pointers have types just like other variables.
- The pointer type must match the type of data you intend to point to.



Example

```
int main()
 {
  int i;       //declare integer
  int *iptr; //declare pointer
  iptr = &i; //initialize pointer
  i = 3;       //initialize i
  return 0;
 }
```

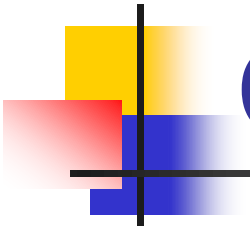


Using the * and & Operators

- The dereferencing operator (*) is used for more than declaring pointers.
- It can also tell the compiler to return the value in the variable being pointed to, rather than the address of the variable.

```
result = *int_ptr;
```

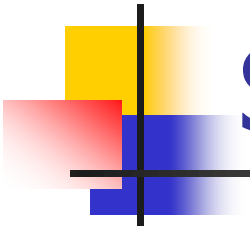
- The variable result is assigned the value of the integer pointed to by `int_ptr`.



Changing Values with *

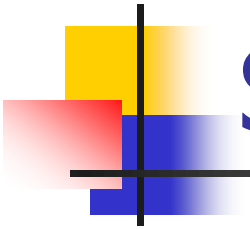
- The dereferencing operator allows you to change the value of the variable the pointer points to.
- For example, the statement below assigns the value 5 to the integer to which int_ptr points.

```
*int_ptr = 5;
```



Summary

- Pointers are variables and constants that hold memory addresses.
- The dereferencing operator (*) is used to declare pointers and to access the value in the variable to which the pointer points.
- The address-of operator (&) returns the address of a variable, rather than the value in the variable.



Summary

- Structures are very useful data structures that allow variables and objects to be grouped to form a new data type.
- Structures are very similar to classes.
- The variables within a structure are called members.

thank you!


QUESTIONS