BENNETT UNIVERSITY

TIMES OF INDIA GROUP

CSET334 - Programming using C++

Module 2: Pointer & Arrays

Course Specific Learning Outcomes

No.	Unit Learning Outcome
CO1	To explain the fundamental programming concepts and methodologies to building C++ programs.
CO2	To implement various OOPs concepts including memory allocation/deallocation procedures and member functions.

List of Contents

- Pointers
  1. Basics
     i. Variable declaration, initialization, NULL pointer
     ii. & (address) operator, * (indirection) operator
     iii. Pointer parameters, return values
     iv. Casting points, void *
  2. Arrays and pointers
     i. 1D array and simple pointer
     ii. Passing as parameter
  3. Dynamic memory allocation
     i. calloc, free, malloc, realloc
     ii. Dynamic 2D array allocation (and non-square arrays)

1. Pointers

- A *pointer* is a reference to another variable (memory location) in a program
- Used to change variables inside a function (reference parameters)
- Used to remember a particular member of a group (such as an array)
- Used in dynamic (on-the-fly) memory allocation (especially of arrays)
- Used in building complex data structures (linked lists, stacks, queues, trees, etc.)

1. Pointers: Visualization

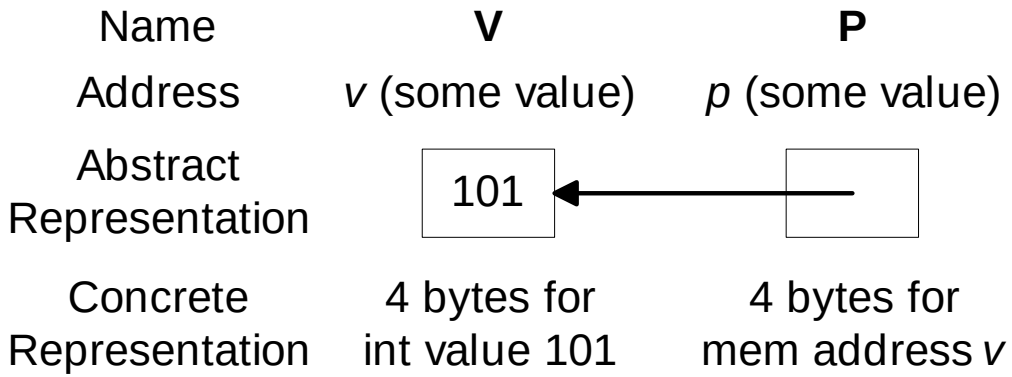
- Variables are allocated at *addresses* in computer memory (address depends on computer/operating system)

- Name of the variable is a reference to that memory address

- A pointer variable contains a representation of an address of another variable (P is a pointer variable in the following):

```
int V = 101;

int *P = &V;
```



1. Pointers: Declaration Syntax

- Basic syntax: *Data_Type *Name*

- Examples:
  - int *P; /* P is var that can point to an int var */
  - float *Q; /* Q is a float pointer */
  - char *R; /* R is a char pointer */

- Complex example:
  - int *AP[5]; /* AP is an array of 5 pointers to ints */

1. Pointers: &(Address of) Operator

- The address (&) operator can be used in front of any variable object in C++
- The result of the operation is the location in memory of the variable
- Syntax: *&Variable*
- Examples:
  - int V;
  - int *P;
  - int A[5];
  - &V - memory location of integer variable V
  - &(A[2]) - memory location of array element 2 in array A
  - &P - memory location of pointer variable P

1. Pointers: Initialization

- **NULL Value Assignment** - Pointer can be initialized with null value.
- Null value assignment is used to indicate pointer points to nothing

- Address Assignment – Pointers can be pointed to other variables addresses using the address (&) op to get address of a variable

- Variable in the address operator must be of the right type for the pointer (an integer pointer points only at integer variables)

- Examples:
  - int V;
  - int *P = &V;
  - int A[5];
  - P = &(A[2]);

1. Pointers: Indirection(*) or Value At Operator

- A pointer variable contains a memory address.
- To refer to the *contents* of the variable that the pointer points to, we use indirection or value at operator.
- Syntax: **pointer_variable_name*
- Example:
  - int V = 101;
  - int *P = &V;
  - /* Then *P would refer to the contents of the variable V (in this case, the integer 101) */
  - cout<<*P /* Prints 101 */

1. Pointers: Examples

```
int A = 3;
int B;
int *P = &A;
int *Q = P;
int *R = &B;

cout<<("Enter value:");
cin>>R;
count<<A<<B;
cout<<*P<<*Q<<*R;
```

```
Q = &B;
if (P == Q)
  cout<<"1";
if (Q == R)
 cout<<"2";
if (*P == *Q)
 cout<<"3";
if (*Q == *R)
 cout<<"4";
if (*P == *R)
 cout<<"5";
```

1. Pointers: Reference as Parameters

- To make changes to a variable that exist after a function ends, we pass the address of (a pointer to) the variable to the function (a reference parameter)

- Then we use indirection operator inside the function to change the value the parameter points to:

```
void changeVar(float *cvar)
 {
  *cvar = *cvar + 10.0;
 }

float X = 5.0;
changeVar(&X);
Cout<<X;
```

1. Pointers: Pointers as Function Return Values

➢A function can also return a pointer value:

```
float *findMax(float A[], int N)
{
  int i;
  float *theMax = &(A[0]);

  for (i = 1;i < N; i++)
    if (A[i] > *theMax)
      theMax = &(A[i]);

  return theMax;
}
```

```
void main()
{
  float A[5] = {0.0, 3.0, 1.5, 2.0,
   4.1};

  float *maxA;

  maxA = findMax(A,5);
  *maxA = *maxA + 1.0;
 cout<<*maxA<<A[4];
}
```

1. Pointers: Pointer to Pointers

- A pointer can also be made to point to a pointer variable (but the pointer must be of a type that allows it to point to a pointer)

- Example:

  - int V = 101;

  - int *P = &V; /* P points to int V */

  - int **Q = &P; /* Q points to int pointer P */

  - cout<<v<<*P<<**Q; /* prints 101 3 times */

1. Pointers: Types

- Pointers are generally of the same size (enough bytes to represent all possible memory addresses), but it is inappropriate to assign an address of one type of variable to a different type of pointer

- Example:

  - int V = 101;
  - float *P = &V; /* Generally results in a Warning in C compiler*/
  - float *P = &V; /* Results in error in C++ compiler*/

- C++ doesn't allow addresses of one data type to be stored in some other data type pointers.

- This makes the C++ programming language strongly typed language.

1. Pointers: Type Casting Pointers

- When assigning a memory address of a variable of one type to a pointer that points to another type it is best to use the cast operator to indicate the cast is intentional (this will remove the error).

- Example:

  - int V = 101;

  - float *P = (float *) &V; /* Casts int address to float */

- Removes error, but is still a somewhat unsafe thing to do

1. Pointers: void Pointer

- A void * is a generic pointer.
- No cast is needed to assign an address to a void * or from a void * to another pointer type
- Example:
  - int V = 101;
  - void *G = &V; /* No error */
  - float *P = G; /* Error*/
- Certain library functions return void * results

1. Pointers: Arithmetic's

- Pointers can be incremented using ++ or – operators
- Example:

```
int a[] = {1, 2, 3,  4, 5};
int *p = a;
p++;
++p;
```

- Integer values can be added and subtracted into the pointers.
- Example:

```
int a[] = {1, 2, 3,  4, 5};
int *p = a;
p = p+1; p--; p = p-1; p--;
```

1. Pointers: Arithmetic's

- Pointers can be subtracted from each other
- Pointer subtraction returns the number of elements between two addresses
- Example:

```
int v[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int * vPtr1 = v[0];
int * vPtr2 = v[2];
then
   vPtr2 – vPtr1 = 2  (i.e. 2 addresses)
```

1. Pointers: Question

➢What is the output of the following C++ Program?

```
#include <iostream>
using namespace std;
int main()
{
      int x = 1, z[2] = {10, 11};
       int *p = NULL;
      p = &x;
      *p = 10;
       p = &z[1];
      *(&z[0] + 1) += 3;
       cout<<x<<z[0]<<z[1];
      return 0;
}
```

GATE CSE 2022

1. Pointers: Question

➢What is the output of the following C++ Program?

```
#include <iostream>
using namespace std;
int main()
{
      int x = 1, z[2] = {10, 11};
       int *p = NULL;
      p = &x;
      *p = 10;
       p = &z[1];
      *(&z[0] + 1) += 3;
       cout<<x<<z[0]<<z[1];
      return 0;
}
```

Answer: 101014

1. Pointers: Question

➢What is the output of the following C++ Program?

```
#include <iostream>
using namespace std;

int main() {
   char c[] = "GATE2011";
   char *p = c;
   cout << (p + p[3] - p[1]);
   return 0;
}
```

GATE CSE 2018

1. Pointers: Question

➢What is the output of the following C++ Program?

```
#include <iostream>
using namespace std;

int main() {
   char c[] = "GATE2011";
   char *p = c;
   cout << (p + p[3] - p[1]);
   return 0;
}
```

Answer: 2011

1. Pointers: Question

➢What is the output of the following C++ Program?

```
void f1(int a, int b) {
 int c;
 c = a; a = b; b = c;
} void f2(int *a, int *b) {
 int c;
 c = *a; *a = *b; *b = c; }
int main() {
 int a = 4, b = 5, c = 6;
 f1(a, b);
 f2(&b, &c);
 cout<< (c - a - b);
 return 0;
}
```

1. Pointers: Question

➢What is the output of the following C++ Program?

```
#include <iostream>
using namespace std;
int main()
{
      int a=20;
      int *ptr=&a;
      int x=a;
      cout<<&*ptr<<endl;
      cout<<&a<<endl;
      return 0;
}
```

GATE CSE 2016

1. Pointers: Question

➢What is the output of the following C++ Program?

```
#include <iostream>
using namespace std;
int main()
{
	int a=20;
	int *ptr=&a;
	int x=a;
	cout<<&*ptr<<endl;
	cout<<&a<<endl;
	return 0;
}
```

Answer: address of a, address of a

1. Pointers: Question

➢What is the output of the following C++ Program?

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
char p[20];
char s[] = "string";
int length = strlen(s);
for (int i = 0; i < length; i++)
        p[i] = s[length - i];
    p[length] = '\0';
    printf("%s\n", p); // Print the result
    return 0;
}
```

1. Pointers: Question

➢What is the output of the following C++ Program?

```
#include<iostream>
using namespace std;
void rer(int **ptr2, int **ptr1)
{
    int *ii;
    ii=*ptr2;
    *ptr2=*ptr1;
    *ptr1=ii;
    **ptr1*=**ptr2;
    **ptr2+=**ptr1;
}
int main()
{
    int var1=5, var2=10;
    int *ptr1=&var1,*ptr2=&var2;
    rer(&ptr1,&ptr2);
    cout<<var2<<endl<<var1;
    return 0;
}
```

ISRO CSE 2020

1. Pointers: Question

➢What is the output of the following C++ Program?

```
#include<iostream>
using namespace std;
void rer(int **ptr2, int **ptr1)
{
    int *ii;
    ii=*ptr2;
    *ptr2=*ptr1;
    *ptr1=ii;
    **ptr1*=**ptr2;
    **ptr2+=**ptr1;
}
int main()
{
    int var1=5, var2=10;
    int *ptr1=&var1,*ptr2=&var2;
    rer(&ptr1,&ptr2);
    cout<<var2<<endl<<var1;
    return 0;
}
```

Answer: 60 50

2. Arrays

- An array is a collection of homogeneous data types stored in contiguous memory locations.

- **Declaration and initialization syntax:** data_type array_name[size] = {values};

- **Example:**

  - int A[5]: A is the address where the array starts (first element), it is equivalent to &(A[0])
  - A is in some sense a pointer to an integer variable
  - To determine the address of A[x] use formula:

    (address of A + x * bytes to represent int)

    (address of array + element num * bytes for element size)

2. Arrays: Using pointer with Arrays

```
float A[6] = {1.0, 2.0, 1.0, 0.5, 3.0, 2.0};
float *theMin = &(A[0]);
float *walker = &(A[1]);

while (walker < &(A[6]))
{
  if (*walker < *theMin)
     theMin = walker;
  walker = walker + 1;
}
cout<<*theMin;
```

2. Arrays: Declaration Examples

➢float B [5]	B is a 1D array of size 5 of floats
➢int * C	C is a pointer to an int
➢char D [6][3]	D is a 2D array of size 6,3 of chars
➢int * E [5]	E is a 1D array of size 5 of pointer to ints
➢int (* F) [5]	F is a pointer to a 1D array of size 5 of ints
➢int G (…)	G is a function returning an int
➢char * H (…)	H is a function returning a pointer to a char

2. Arrays: Using 1D Arrays as Parameters

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

- When passing whole 1D array as parameter to a function, we use the syntax: *data_type paramName[]*, but we can also use *data_type *param_name*

```
int totalArray(int*A, int N)
{
  int total = 0;
  for (i = 0;i < N; i++)
    total += A[i];
  return total;
}
```

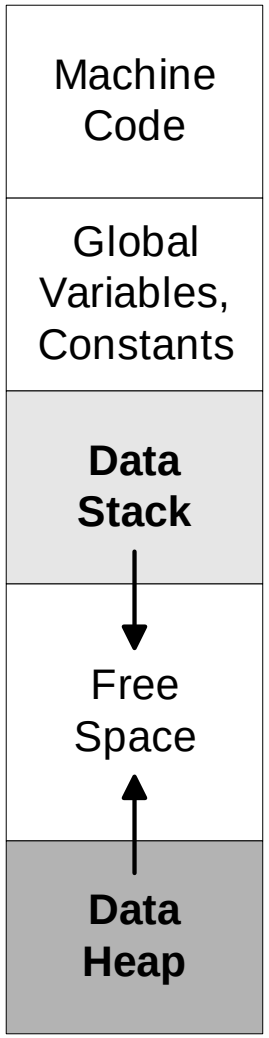
- For multi-dimensional arrays we still have to use the:
  data_type ArrayName[Dim1][Dim2][Dim3] … form.

3. Dynamic Memory Allocation

- Space for program code includes space for machine language code and data

- Data is divided into:
  - Global Space: Space for global variables and constants
  - Data Stack - expands/shrinks while program runs
  - Data Heap - expands/shrinks while program runs

- Local variables in functions are allocated when function starts:
  - Local variables are placed on the data stack
  - When function ends, space is freed up
  - Static allocation needs to know the size of the data item during compilation



3. Dynamic Memory Allocation

- Allows the program to dynamically create new variables, arrays and objects on the heap data segment during the program execution

- C uses the malloc() and calloc() function to allocate memory dynamically at run time and uses a free() function to free dynamically allocated memory.

- In C++, the dynamic memory allocation can be implemented using the following two operators:

  i. new operator
  ii. delete operator

3. Dynamic Memory Allocation: new Operator

- The new operator denotes a request for memory allocation on the Free Store of Heap
- If sufficient memory is available, a new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.
- Data type could be any built-in data type including array or any user-defined data type including structure and class.

- Syntax for using new operator with built in data type:

pointer-variable = new data-type;

- For custom data types, a constructor is required (with the data type as input) for initializing the value. e.g: *pointer-variable = new data-type(value);*

3. Dynamic Memory Allocation: new Operator Example

```
#include <iostream>
#include <memory>
using namespace std;

int main()
{
    // pointer to store the address returned by the new
    int* ptr;
    // allocating memory for integer
    ptr = new int;

    // assigning value using dereference operator
    *ptr = 10;

    // printing value and address
    cout << "Address: " << ptr << endl;
    cout << "Value: " << *ptr;

    return 0;
}
```

3. Dynamic Memory Allocation: new Operator Example

```cpp
#include <iostream>
#include <string>

using namespace std;

// Define a structure with a constructor
struct Person {
    string name;
    int age;

    // Constructor to initialize the members
    Person(const string& personName, int personAge) {
        name = personName;
        age = personAge;
    }
};

int main() {
    // Dynamically allocate memory for a Person structure using the constructor
    Person* person = new Person("Rakesh Kumar", 25);

    // Display the values
    cout << "Name: " << person->name << endl;
    cout << "Age: " << person->age << endl;

    // Free the allocated memory
    delete person;

    return 0;
}
```

3. Dynamic Memory Allocation: Common Issues

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

1. Memory Leaks

```
int* ptr = new int;
*ptr = 10;
// No delete -> Memory leak
```

2. Dangling Pointers

```
int* ptr = new int(5);
delete ptr;
cout << *ptr;  // Undefined behavior
```

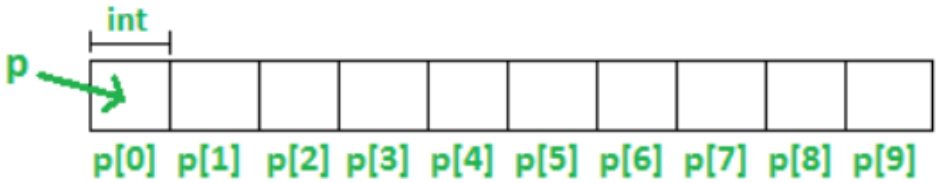
3. Dynamic Memory Allocation: new Operator

- A new operator is also used to allocate a block(an array) of memory of type data type.

- Syntax: *pointer-variable = new data-type[size];*

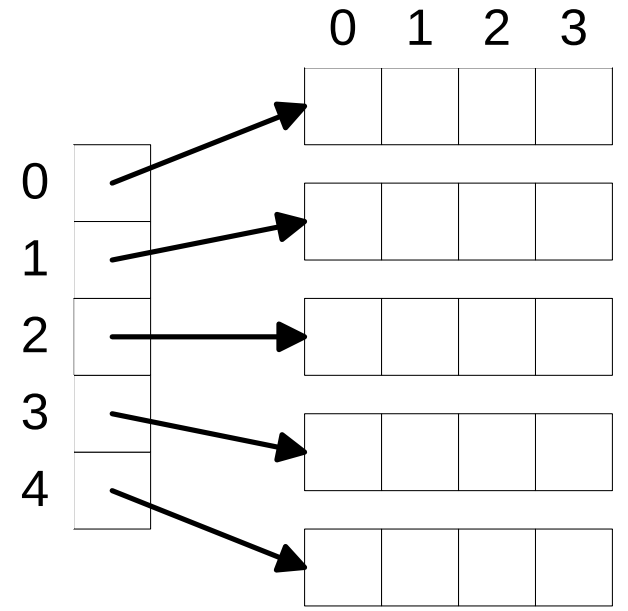
- Example: *int *p = new int[10];*

- Visualization



3. Dynamic Memory Allocation: Dynamic Arrays

- By default, C++ does not support 2D arrays with unequal row sizes.
- To solve this issue, we use dynamic memory allocation to create 2D arrays with rows of varying sizes.
- Create multiple 1D arrays of different sizes dynamically.
- Store the starting pointer of each 1D array in an array of pointers.
- This approach allows access to each 1D array through the array of pointers, effectively enabling the use of 2D arrays with unequal row sizes.



3. Dynamic Arrays: Example

```
#include <iostream>
using namespace std;

int main() {
    // Step 1: Create an array of pointers (for the rows)
    int rows;
    cout << "Enter the number of rows: ";
    cin >> rows;

    int** dynamicArray = new int*[rows];
```

```
// Step 2: Create each row with a different size
for (int i = 0; i < rows; ++i) {
    int size;
    cout << "Enter the size of row " << i + 1 << ": ";
    cin >> size;

    dynamicArray[i] = new int[size];

    // Initialize the row with values
    cout << "Enter " << size << " elements for row " << i + 1 << ": ";
    for (int j = 0; j < size; ++j) {
        cin >> dynamicArray[i][j];
```

```
// Step 3: Display the dynamic array
cout << "\nDynamic Array Contents:\n";
for (int i = 0; i < rows; ++i) {
    cout << "Row " << i + 1 << ": ";
    for (int j = 0; dynamicArray[i][j] != '\0' && j < sizeof(dynamicArray[i])/sizeof(int);
        cout << dynamicArray[i][j] << " ";
    }

}
```

```
// Step 4: Free the allocated memory
for (int i = 0; i < rows; ++i) {
    delete[] dynamicArray[i];
}
delete[] dynamicArray;

return 0;
```

3. Dynamic Memory Allocation: delete Operator

- delete is an operator that is used to destroy array and non-array(pointer) objects which are dynamically created by the new operator.
- The new operator is used for dynamic memory allocation which stores variables on heap memory.
- This means the delete operator deallocates memory from the heap.
- The pointer to the object is not destroyed, the value or memory block pointed by the pointer is destroyed.
- The delete operator has void return type which means it does not return any value.

3. delete Operator: Example

```
// Program to illustrate deletion of array

#include <bits/stdc++.h>
using namespace std;

int main()
{
    // Allocate Heap memory
    int* array = new int[10];

    // Deallocate Heap memory
    delete[] array;

    return 0;
}
```

```
// C++ program for deleting
// NULLL pointer

#include <bits/stdc++.h>
using namespace std;

int main()
{
    // ptr is NULL pointer
    int* ptr = NULL;

    // deleting ptr
    delete ptr;

    return 0;
}
```


THANK YOU!