Date

Programming with C++ pointers

A pointer is a variable that holds a memory address usually location of another variable in memory.

Importance of pointers in C++

1. Memory location of variable can be directly accessed.
2. Supports C++ dynamic allocation routines.
3. Improve efficiency of certain routines.

C++ Memory map.

1. Program code → this is the area where compiled code is saved.
2. Global variable → It stores the global variables.
3. Stack → To save function call return, arguments, local variables.
4. Heap → It is used for dynamic allocatio

Date

Programming with C++ pointers

A pointer is a variable that holds a memory address usually location of another variable in memory.

Importance of pointers in C++

1. Memory location of variable can be directly accessed.
2. Supports C++ dynamic allocation routines.
3. Improve efficiency of certain routines.

C++ Memory map.

1. Program code → this is the area where compiled code is saved.
2. Global variable → It stores the global variables.
3. Stack → To save function call return, arguments, local variables.
4. Heap → It is used for dynamic allocatio

Date

Memory allocation : ① Static
② Dynamic.

Static memory allocation : Memory to be allocated is known beforehand and memory is allocated during compilation itself.

```
short i;   // 2 bytes of internal memory
           allocated to i;
```

Dynamic memory allocation : Memory to be allocated is not known beforehand. Memory is allocated during runtime.

Operator for Dynamic Allocation.

New
↳ Allocates dynamic memory.
↳ Returns the pointer storing memory address.

Delete
↳ Deallocates the memory.

function, pure virtual function.

Date

Declaration and Initialization of Pointers

General form :

Type *variablename ;

↓ ↓

DataType pointer name.

eg. int *P1 ;

pointer to integer.

float *f1 ;

pointer to float

char *c1

pointer to character.

Date

Program 1

```
#include <iostream.h>
int main()
{   int i = 10;          → variable of type int
    int *ptr;    // declaring a pointer of
                    type int
    ptr = &i;    // storing the address of i in
                    ptr
                    i.e address of i gets stored
                    ptr.
cout << " Address of Integer variable i is:"
     << &i << endl;
cout << " Address of Pointer variable ptr is:"
     << ptr << endl;
cout << " Value of Pointer variable i is:"
     << *ptr << endl;
                  ↳ value at address pointed
                    by pointer.
return 0;
}
```

Date

Output: Address of Integer variable is :
0x69fef8

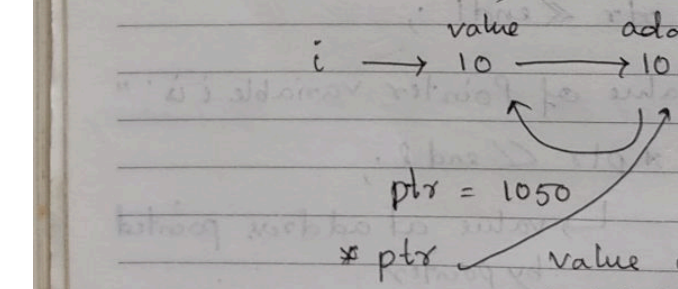
" " Pointer variable ptr is :
0x 69fef8.

value of pointer variable i is : 10.

& with pointer : This operator returns memory address of the operand.

* with pointer : changes or accesses the value being pointed to by pointer.

example. int i = 10;
int *ptr;
ptr = &i;



∴ *ptr = 10;

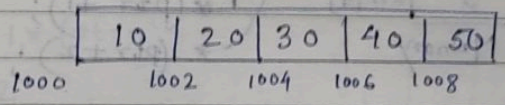
Date

Pointer Arithmetic :

Only addition and Subtraction may be performed on pointers [note: only integer can be added or subtracted from pointers]

[When pointer is increamented by 1, it points to the memory location of the next element of its base type]

example :

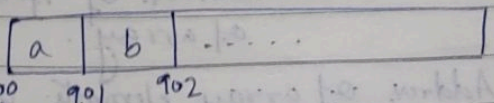


int *ptr ; { ptr is pointing to 1000 initially }

ptr ++ ; { ptr = ptr + 1 } → 1001 ✗ This is wrong.

↳ 1002 ✓ next address.

ptr = ptr + 2 ; // 1004 → ptr - - ; 1002



char *c1 ;

c1++ ;

c1 = c1 + 2 ;

INT = 2 Bytes.

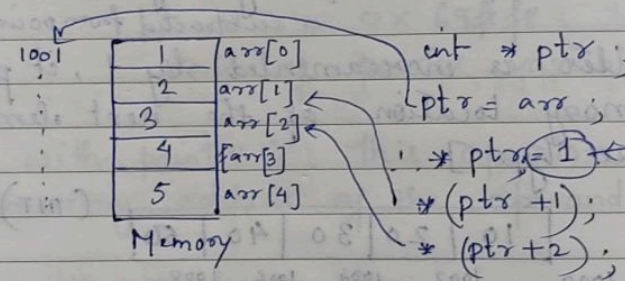
char = 1 Byte.

Float = 4 Bytes.

Date

Arrays using pointers :

arr[5] = {1, 2, 3, 4, 5};



Array using pointers

```
#include <iostream.h>
int main()
{  int arr[5] = {1, 2, 3, 4, 5}
   int *ptr; // Pointer
   ptr = arr; // Now ptr contains the
                 address of first element
                 of array.
cout<< "Address of array elements using
        pointers" << endl;
for (int i=0; i<5; i++)
```

Date

```
{ cout<<"value of element"<<i<<" = "<<*(ptr+i)
    <<" and Address is:"<<(ptr+i)<<endl;
}
return 0;
}
```

i = 0 , *ptr = 1

i = 1, *(ptr+1) = 2

i = 2

⋮

output: Address of array elements using pointers.

value of element : 0 = 1 and Address is : 0x69fed4

" " : 1 = 2 " " : 0x69fed8

" " : 2 = 3 " " : 0x69fedc

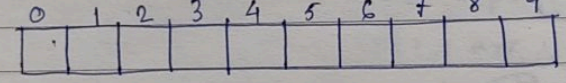
" " : 3 = 4 " " : 0x69fee0.

" " : 4 = 5 " " : 0x69fee4.

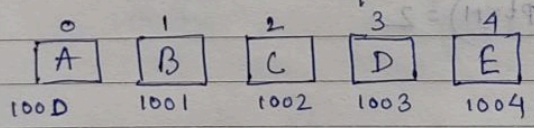
Date

Array of Pointers :

It is used to declare pointers as an array.
for eg., int *i[10]., is an array holding 10 int pointers



char * arr[5] = {"A", "B", "C", "D", "E"};



* arr[1] = B
& arr[1] = 1001

example:

```
#include <iostream.h>
int main()
{ char * arr[5] = {"A", "B", "C", "D", "E"};
  for (int i=0; i<5; i++)
  { cout << *arr[i] << endl;
  }
  return 0;
}
```

o/p
* arr[0];
* arr[1];
* arr[2];

or if

& arr[i]; → o/p → points the address.

Date

o/p ⇒ A
B
C
D
E

Pointers & Functions :

A function may be invoked in one of the two ways :
① call by value → actual value is passed
② call by reference.
↳ passing reference.
↳ passing pointer.

Call by value :

```
void m1( int i);
int main()
{ int i = 5;
  m1(i);
  cout << i;
  return 0;
}
void m1(int i)
{ i = 10; // i is now 10.
}
```

value is passed.

5 → i is local to m1, so value of i is not changed in main().

o/p :
5
10 ✗

Date

```
void m1(int i)
{
    i=10;
}
```

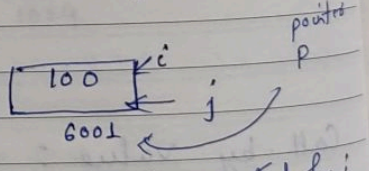
Call By Reference.

- Reference.
  - It is an alias name for a variable
- Pointer
  - Holds the memory address of variable.

```
int i=100;
int &j=i;
```

reference → `int &j=i;` ← alias name

No separate memory is created for j.




```
cout << i;   // 100
cout << j;   // 100
```

O/P

Date

pointer

```
int *p;
p=&i ;  // address of i is stored in pointer p.
cout << *p;   [O/P → 100]
```

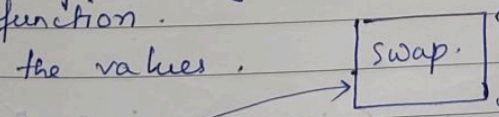
Example of invoking functions by Passing reference:

In this the parameters are passed to functions by reference. So called function doesn't create its own copy of original values, rather it refers to original value by different names (i.e. reference).

[Any function changes reflected in original data]

eg. values of the original variables are to be changed by function.

Swap the values.



x=10
y=5

```
cout <<x;   // 5
cout <<y;   // 10
```

O/P.

it takes

```
#include <iostream.h>
void swapnumbers (int& , int&);   // function prototype.
                  ↑ reference variable as input

int main()
{
  int i=5 , j=10 ;   // variable declaration.
  cout << "values of i & j before swapping" <<endl;
  cout << "i=" <<i <<endl ;
  cout << "j=" <<j <<endl ;
  cout << " start swapping the values " <<endl;
  swap (i, j);
  cout << "values of i & j after swapping" <<endl;
  cout << "i=" << i <<endl;
  cout << "j=" << j <<endl;
  return 0;
}
void swapnumbers (int &n1, int &n2)
                  ↳ alias of i   ↳ alias of j
  { int temp;
    temp= n1;
    n1=n2;
    n2=temp;
  }
```

i = 10
j = 5

Date

o/p =>

values of i & j before swapping

i = 5

j = 10.

Start swapping the values.

value of i & j after swapping

i = 10

j = 5.

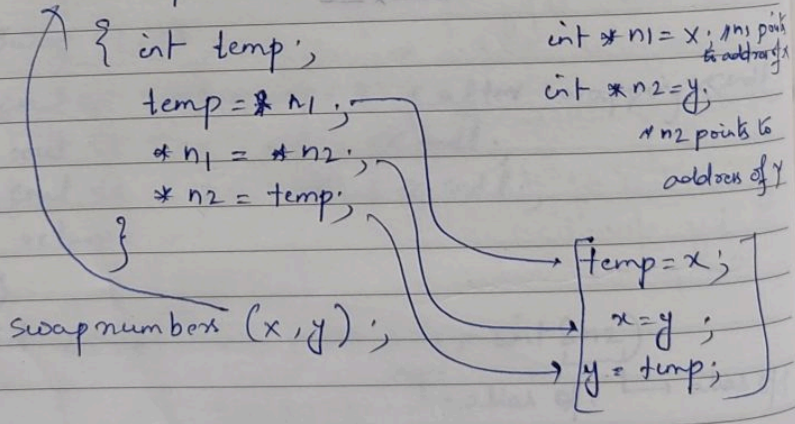
—x—

Date

Example of invoking functions by passing parameters.

In this address of actual arguments in the calling function are copied into formal arguments of the called function. Called function doesn't create its own copy of original values.
[changes are reflected in original data]

eg. void swapnumbers (int *n1, int *n2)



Date

```
#include <iostream.h>
void swapnumbers (int *, int *); // function prototype.
int main ()
{ int i= 20 , j=40 ;
  cout << " value of i & j before swapping "<<endl;
  cout << "i = " <<i <<endl;
  cout << "j= " <<j <<endl;
  cout <<"start swapping the values "<<endl;
  swap (i, j);
  cout << " values of i & j afterswapping "<<endl;
  cout << "i= " << i << endl;
  cout << "j= " << j << endl;
  return 0;
}
void swapnumbers (int *n1, int *n2)
{ int temp;
  tem = *n1;
  *n1 = n2;
  *n2 = temp;
}
int *n1 = i;
int *n2 = j;
```

Date

o/p: value of i & j before swapping.
i = 20
j = 40.
start swapping the values. Value of i & j after swapping.
i = 40
j = 20.

—x—

Functions:
example of function returning Reference:
In this function returns a reference.

eg. int & max (int &a, int &b)
10 5
{ if (a > b)
return a;
else return b;
}
returns reference to either variable a or b depending on the condition.

(max (x, y) = 1;
x = 10
Y = 5

Date

Program : Function Returning Reference.

```
#include <iostream.h>
int num;    // Global variable.
int & func1(); // function Prototype // this function returns the reference.

int main()
{  func1() = 20;  ⟶  num = 20;
   cout << num;
   return 0;
}
int & func1()
{  return num;
}
```

o/p = 20.

Date

Function Returning Pointers (example)

Syntax of a function returning a pointer, would be,

type * function name (argument list);

eg. int *maxnumber (int &, int&);

↳ the memory address of a variable is returned back.

```
int *c;
c = maxnumber( );
```

Program: Function Returning Pointers.

```
#include <iostream.h>
int num;        // Global variable.
int * maxnumber (int &, int&); // Function prototype

int main ()
  { int num1, num2, *ptr;
    cout << "Enter the numbers "<<endl;
    cin >> num1>>num2;
```

Date

```
ptr = maxnumber (num1, num2);
cout << "Bigger number is =" << *ptr;
return 0;
}
int * maxnumber (int &x, int &y)
{ if (x>y)  if (x>5)
  { return (&x);
  }
else
  { return (&y);
  }
}
```

num1 → x
num2 → y

ptr = memory address of variable "num2"

O/P: → Enter the numbers.
4
5
num1=4, num2=5
Bigger number is = 5

Date

This Pointer :

This is a keyword = current instance of class.

↳ stores the address of current calling object.

Uses :

(i) It is used to pass current object as parameter.

(ii) It is used to refer current class instance variable.

```
class ABC
{ int val; // instance variable.
  public: abc(int val)
  {
    val = val; X this will print ambiguous result so wrong.
    this -> val = val.
  }
  void display()
  { cout << val;
    cout << this -> val;
    cout << this; → print the addrs of current object
```

Date

This Pointer Program

```
#include <iostream.h>
class student
  { public : int sid ;
             string sname ;
             student (int sid , string sname)
             { this -> sid = sid ,
               this -> sname = sname ;
             }
             void display ()
             { cout << sid << " " << sname << endl;
             }
  };
int main ()
  { student S1 = student (101 , "Aman");
    Student S2 = student (102 , " Mohan");
     S1 . display ();
     S2 . display ();
     return 0;
  }
```

constructor

S1->sid = 101
S2->sname = Aman
S2->sid = 102
S2->sname = Mohan

O/P

101 Aman
102 Mohan.

Polymorphism, Categorization of polymorphism techniques: compile tim
by parameter, run time polymorphism pointers to derived class , Early bindi
function, pure virtual function.