Binding-Virtual Functions-Generic Programming with Templates-Application with Files

UNIT-5

Dr. BALAMURUGAN
ASST. PROFESSOR
DEPARTMENT OF COMPUTER SCIENCE
GOVERNMENT ARTS COLLEGE
COIMBATORE
EMAIL: spbalamurugan@rediffmail.com

INTRODUCTION

What is Binding?

- **Binding** refers to the process of converting identifiers (such as variable and performance names) into addresses.
- **Binding** is done for each variable and functions.
- For functions, it means that matching the call with the right function definition by the compiler.
- It takes place either at compile time or at runtime.
- Types of binding: Late binding & Early binding

INTRODUCTION -BINDING

- **Early Binding (Compile-time time polymorphism)** - As the name indicates, compiler (or linker) directly associate an address to the function call.
- It replaces the call with a machine language instruction that tells the machine to jump to the address of the function.
- By default early binding happens in C++.
- **Late Binding : (Run time polymorphism)**, the compiler adds code that identifies the kind of object at runtime then matches the call with the right function definition.
- Late binding is achieved with the help of virtual keyword.

EARLY BINDING - EXAMPLE

```
class Base
{
public:
   void show() { cout<<" In Base \n"; }
};

class Derived: public Base
{
public:
   void show() { cout<<"In Derived \n"; }
};
 int main(void)
{
   Base *bp = new Derived;
     // The function call decided at  compile time (compiler sees type of pointer and calls base class
   function.
   bp->show();
   return 0;
}
```

LATE BINDING - EXAMPLE

```
class Base
{
public:
  virtual void show() { cout<<" In Base \n"; }
};

class Derived: public Base
{
public:
   void show() { cout<<"In Derived \n"; }
};
 int main(void)
{
   Base *bp = new Derived;
    // The function call decided at  compile time (compiler sees type of pointer and calls base class
   function.
   bp->show();
   return 0;
}
```

WHAT IS A POINTER?

- A pointer is a variable that holds the memory address of another variable of same type.
- It is created with the * operator.
- In pointer * is called as dereference operator.
- It supports dynamic memory allocation routines.
- Pointer declaration and initialization - Syntax:

```
datatype *pointer_variable_name;
datatype variable_name;
*pointer_variable_name=&variable_name;
```

- Pointer declaration and initialization - Example:

```
int *ptr;  int x;  *ptr=&x;
```

- Get the memory address of the variable using address operator(&) called reference operator. Read the value of variable using (*) called dereference operator.

POINTER ARITHMETIC OPERATIONS

- For e.g. **int *x; x++;**
- If current address of x is 1000, then x++ statement will increase x by 2(size of int data type) and makes it 1002, not 1001.
- Exercise :

```
int x=10;
int *ptr=&x; //Assume the current address of x is 1000.
cout<<*(ptr++)<<*(++ptr)<<++(*ptr)
```

POINTERS AND ARRAYS

- C++ treats the name of an array as constant pointer which contains base address i.e. address of first location of array.
- For eg. **int x[10];**
- Here x is a constant pointer which contains the base address of the array x.
- We can also store the base address of the array in a pointer variable.
- It can be used to access elements of array, because array is a continuous block of same memory locations.

For eg. **int x[5];**

int * ptr=x; // ptr will contain the base address of x

(OR)

int * ptr= &x[0]; //ptr will contain the base address of x

POINTERS AND ARRAYS - EXAMPLE

```
int main ()   {
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
double *p;
p = balance; // output each array element's value
cout << "Array values using pointer " << endl;
for ( int i = 0; i < 5; i++ ) {
cout << "*(p + " << i << ") : ";
cout << *(p + i) << endl;
}
cout << "Array values using balance as address " << endl;
for ( int i = 0; i < 5; i++ ) {
cout << "*(balance + " << i << ") : ";
cout << *(balance + i) << endl;
 }
 return 0;
 }
```

POINTERS AND STRINGS

- String is a character array or array of character.
- Pointer can be used for handling the character array or String.
- Example :

```
void main()
{
    char str[] = “computer”;
    char *cp=str;
    cout<<str; // using variable name
    cout << cp; // using pointer variable
}
```

Here ‘cp’ stores the address of str. The statement **cout<<cp;** will print computer as giving an address of a character with cout results in printing everything from that character to the first null character that follows it.

ARRAY OF POINTERS

- Like an array, we can also have an array of pointers.
- A common use of array of pointers is an array of pointers to strings.
- An array of character pointers is used for storing several strings in memory.
- Example :

```
char * vehicle[ ]={“CAR”,”VAN”,”CYCLE”,”TRUCK”,”BUS”};
for(int i=0;i<5;i++)
cout<<vehicle[i]<<endl;
```

- In the above example, the array vehicle[] is an array of character pointers. Thus vehicle[0] contains the base address of the string “CAR”, vehicle[1] contains the base address of the string “VAN” and so on.

References

- https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_model.htm
- https://www.geeksforgeeks.org/object-oriented-programming-in-cpp
- https://www.javatpoint.com/cpp-oops-concepts

TEXT BOOKS

1) Grady Booch, “Object Oriented Analysis and Design with Applications”, Second Edition, Pearson Education.

2) Ashok N. Kamthane, “Object Oriented Programming with ANSI &Turbo C++” , First Indian Print, Pearson Education, 2003.

REFERENCE BOOKS

1) Balagurusamy “Object Oriented Programming with C++”, TMCH, Second Edition, 2003.