overloading.md

Finding Gross Pay

Three are three types of employees in Indian railways. They are regular, daily wages and consolidated employees. Gross Pay for the employees are calculated as follows:

- regular employees - basic + hra + % of DA * basic
- Daily wages – wages per hour * number of hours
- Consolidated – fixed amount

Overloading

- Overloading is the reuse of the same function name or symbol for two or more distinct functions or operations.
- There are 2 types of Overloading in C++ :
  i. Function Overloading -> A function can be declared more than once which makes its meaning different.
  ii. Operator Overloading -> An operator can be declared more than once with different set of operations.

Example of Function Overloading

```
#include<iostream>
void test (int x, int y);
void test (char ch);
int main()
{
int n1 = 10;
int n2 = 30;
char ch = ‘*’;
Test (n1,n2);
Test (ch);
return (0);
}
void test (int x, int y)
{
    cout<<x+y;
}
void test (char ch)
{
    cout<<ch;
}
```

Signature of a Function

- A function’s argument list (i.e., number and type of argument) is known as the function’s signature.
- Functions with Same signature - Two functions with same number and types of arguments in same order.
- Variable names doesn’t matter. For instance, following two functions have same signature.

```
void squar (int a, float b);    //function 1
void squar (int x, float y);
```

Function Overloading

- C++ enables several functions of the same name to be defined, as long as they have "different signatures."
- The C++ compiler selects the proper function to call by examining the number, types and order of the arguments in the call.
- Overloaded functions are distinguished by their signatures
- C++ compilers encodes each function identifier with the number and types of its parameters (sometimes referred to as name mangling or name decoration) to enable type-safe linkage.

Code Example

- Following code fragment overloads a function name prnsqr( ).

```
void prnsqr (int i);        //overloaded for integer #1
void prnsqr (char c);   //overloaded for character #2
void prnsqr (float f);  //overloaded for floats #3
void prnsqr (double d); //overloaded for double floats #4
void prnsqr (int i)
{
cout<<“Integer”<<i<<“’s square is”<<i*i<<“\n”;
}
void prnsqr (char c);
{
cout <<“No Square for characters”<<“\n”;
}
void prnsqr (float f)
{
cout<<“float”<<f <<“’s square is”<<f *f<<“\n”;
}
void prnsqr (double d)
{
cout <<“Double float”<<d<<“’s square is”<<d*d<<“\n’;
}
```

Resolution by Compiler when it sees second function with same name

- Signatures same with same return types -> ERROR
- Signatures same with different return types -> ERROR

  Eg :

```
 float square (float f);
 double square (float x);
// Differ only by return type so erroneous re-declaration
```

- Signatures different -> considered to be OVERLOADED.

Steps Involved in Finding the Best Match for a function call

1. **One match**

   A match is found for the function call.

   Eg :

   ```
   void afunc(int);
   void afunc(double);
   afunc(0);
   //The function call is matched to void afunc(int); and compiler invokes corresponding function definition as 0 (
   ```

- **A match through Promotion -**

  If no exact match is found, an attempt is made to achieve a match through promotion of the actual argument. ### Integral promotion

  - Conversion of integer types (char, short, etc.) into int. ### Eg :

    ```
      void afunc (int);
      void afunc (float);
      afunc (‘c’);
    // This will invoke afunc(int)
    ```

  ### Standard Conversion

  - Normal conversion of integer to other types.

    Eg :

    ```
    void afunc (char);
    afunc (96);
    // This will invoke afunc(char)
    ```

- **Ambiguous Match**

  More than one defined instance for the function call.

  ### Eg :

  ```
  void afunc(char);
  void afunc(double);
  // Compiler will be confused as to pick which out of char, double.
  // Results in ERROR.
  ```

  # Default Arguments Versus Overloading

  - Using default argument is also overloading, because the function may be called with an optional number of arguments. ### Eg :

  ```float amount (float principal, int time=2, float rate=0.08);

  ```
  Now this function may be called by providing just one or two or all three argument values. A function call like
        cout<<amount (3000);
  will invoke the function amount() with argument values 3000, 2, and 0.08 respectively. Similarly a function call
      cout <<amount (3000,4);
  Will invoke amount() with argument values 3000, 4 and 0.08
      cout <<amount (2500,5,0.12);
  Will invoke amount() with argument values 2500, 5, and 0.12 respectively

  # Solution to the "Find Gross Pay" problem
  - Refer to program codes.

  ## THANK YOU
  ```
  ```