LABORATORIO II
ICOM 4015 ADVANCED PROGRAMMING

a. Reconocer la importancia de identificar cuando se deben utilizar parámetros por valor y cuando por referencia.
b. Argumentos funcionales. ¿Por qué se deben utilizar?.

Debemos tener en cuenta, que existen tres diferentes formas de pasar argumentos o variables a una función. Pero a veces no es aparente cuando se debe utilizar uno u otro modo. La siguiente es una forma sencilla de recordar cuando se puede hacer uso de cada una:

Por valor.

```
int max(int i, int j);
```

¿Cuándo no es apropiado introducir parámetros por valor?

- Objetos muy grandes no deben pasarse como argumentos por valor. Puesto que al ser copiados ocupan demasiado espacio de memoria y a su vez desperdicia tiempo;

- Si los valores de los argumentos necesitan ser modificados dentro de la función. La acción de pasar por valor, crea copias locales de un argumento, dejando los argumentos por fuera intactos.

¿Cuándo es apropiado introducir parámetros por valor?

- El argumento es de tipo fundamental y, además es pequeño;

- Si no deseamos modificar tales argumentos.

Por puntero. (El tema relacionado con punteros se verá más adelante en el curso)

```
int max(int *i, int *j);
```

Por referencia.

```
int max(int &i, int &j);
```

¿Cuándo no es apropiado introducir parámetros por referencia?

- El parámetro debe referir a diferentes objetos dentro de una función.

¿Cuándo es apropiado introducir parámetros por referencia?

- Los argumentos necesitan ser cambiados dentro de la función;
- El valor necesita ser vuelto desde la función;
- Si los parámetros son objetos complejos y grandes.

El uso de los parámetros por referencia permite que una función cambie el valor del parámetro actual o argumento, el cual es usado cuando se llama la función. Por defecto, el método utilizado es el de pasar argumentos por valor, lo cual implica que la función invocada no puede cambiar el parámetro real. Esto permite tener una cierta protección de los datos originales. Utilizamos solamente variables por referencia cuando deseamos cambiar el valor de un parámetro real y además permite ahorrar un poco de memoria en el proceso.

2. Ejemplos

a. Cálculo del promedio ponderado de un estudiante.

```
#include <iostream.h>
#include <iomanip.h>

// Auxiliary functions
void readAssignmentGrades(float& assignment1, float& assignment2);
void readExamGrades(float& ex1, float& ex2);
void readFinalGrade(float& final);
float calculateAverage(float assignment1, float assignment2, float exam1, float
exam2, float finalExam);

void printReport(float assignment1, float assignment2, float exam1, float exam2,
float finalExam, float average);

void ReadFloat(int l1, int l2, float& f);

int main() {
  float assignment1, assignment2;
  float exam1, exam2;
  float finalExam;

  cout << "Programa que computa promedio ponderado de un curso para un
estudiante. " << endl
       << "Notas incluyen dos asignaciones, dos examenes parciales y un examen
final." << endl
       << "Todas las notas son entradas por el teclado." << endl << endl;

  readAssignmentGrades(assignment1, assignment2);
  readExamGrades(exam1, exam2);
  readFinalGrade(finalExam);
  float avg;
  avg = calculateAverage(assignment1, assignment2, exam1, exam2, finalExam);
  printReport(assignment1, assignment2, exam1, exam2, finalExam, avg);
  return 0;
}

// Auxiliary functions

void readAssignmentGrades(float& assignment1, float& assignment2) {
  cout << "Asignacion1 -> ";
  ReadFloat(0,100,assignment1);
  cout << "Asignacion2 -> ";
  ReadFloat(0,100,assignment2);
}

void readExamGrades(float& ex1, float& ex2) {
  cout << "Examen1 -> ";
  ReadFloat(0,100,ex1);
  cout << "Examen2 -> ";
  ReadFloat(0,100,ex2);
}

void readFinalGrade(float& final) {
  cout << "Examen Final -> ";
  ReadFloat(0,100,final);
}

float calculateAverage(float assignment1, float assignment2,
                       float exam1, float exam2,
                       float finalExam)
{
  // Calculate assignments average
  // Calculate exams average
  // Calculate weighted average
return ((assignment1 + assignment2 + exam1 + exam2 + finalExam) / 5);
}
```

```
void printReport(float assignment1, float assignment2, float exam1, float exam2,
float finalExam, float average)
{
  cout << setprecision(4) << endl;
  // print assignment grades
      cout << "Assignments: " << assignment1 << ", " << assignment2 << endl;
  // print exam grades
      cout << "Exams: " << exam1 << ", " << exam2 << endl;
  // print final exam grades
      cout << "Final exam: " << finalExam << endl;
  // print weighted average
      cout << endl << "Average: " << average << endl;
}

void ReadFloat(int l1, int l2, float& f)
{
      float temp;
      bool good = false;

      do {
              cout << "Entre valor dentro de ["
                     << l1 << "," << l2 << "]" << endl;
              cin >> temp;
              if (temp >= l1 && temp <= l2) good = true;
              else cout << "Valor no esta dentro del rango ["
                              << l1 << "," << l2 << "]" << endl;
      } while (!good);
      f = temp;
}
```

b. **Integral de una función en un intervalo dado [a, b].**

```
#include <iostream>

// Forward definitions
double integrate(double a, double b, double n, double f(double x));
double cube(double x);
double sqr(double x);


int main() {
  cout << "Integral of x^2 in [0,1] = "
       << integrate(0.0, 1.0, 10000, sqr)
       << endl;
  cout << "Integral of x^3 in [0,1] = "
       << integrate(0.0, 1.0, 10000, cube)
       << endl;
}

double integrate(double a, double b, double n, double f(double x)) {
  double delta = (b-a) / double(n);
  double sum = 0.0;
  for (int i=0; i<n; i++) {
    sum += f(a + delta * i) * delta;
  } return sum;
}

double cube(double x) {
  return x * x * x;
}

double sqr(double x) {
  return x * x;
}
```

c. Aplicación del Método Newton-Rhpason para hallar raíces de funciones.

```
#include <math.h>
#include <iostream.h>

double f(double p);
double f1(double p);
double Pi(double g, double g1);
bool NewtonRaphsonRoot(double &p, double TOL, long N, double g(double p), double
g1(double p));


int main() {
  double P;
  long N;
  double tolerancia;

  cout << "Este programa resuelve para dar una raiz de f(x) = 0 de una funcion"
<< endl
       << "diferenciable a partir de una aproximacion inicial P.  Se entra la"
<< endl
       << "Tolerancia, y el numero maximo de iteraciones." << endl << endl;
  cout << "Entre P subzero: ";
  cin >> P;
  cout << "Entre la tolerancia: ";
  cin >> tolerancia;
  cout << "Entre el numero de iteraciones: ";
  cin >> N;

  if (NewtonRaphsonRoot(P, tolerancia, N, f, f1))
    cout << "Aproximacion resultante: " << P << endl;
  else cout << "Aproximacion no fue encontrada por que la tolerancia no fue
alcanzada" << endl
         << "antes de finalizar el numero maximo de iteraciones." << endl <<
endl;

  return 0;
}


bool NewtonRaphsonRoot(double& p, double TOL, long N, double g(double p), double
g1(double p)) {
  double p0;
  p0 = p;
  for (long i = 1; i <= N; i++) {
    p = p0 - Pi(g(p0),g1(p0));
    if (fabs(p - p0) < TOL) return true;
    else p0 = p;
  }
  return false;
}

double Pi(double g, double g1) {
  return g/g1;
}

double f(double p) {
  return (pow(p,3) - 3 * p + 12);
}

double f1(double p) {
  return (3 * pow(p,2) -3);
}
```