UNIVERSIDAD TECNOLÓGICA
DEL CENTRO DE VERACRUZ

Reporte Final de Estadía

Jesús David Sandoval Romero

Verificador de entradas y salidas de personal.

Universidad Tecnológica del Centro de Veracruz

Programa Educativo de Ingeniería en Mantenimiento Industrial

Reporte que para obtener su título de Ingeniero en Mantenimiento Industrial

Proyecto de estadía realizado en la empresa:

Nombre de la empresa

H. Ayuntamiento constitucional de Ixhuatlancillo Ver.

Nombre del Asesor Industrial:

Ing. Dani Espinoza Reyes

Nombre del Asesor Académico:

Ing. Rafael Martínez Meneses

Cuitláhuac, Ver., a 13 de Abril de 2018

Contenido

RESUMEN

El presente documento describe todo lo referente en el desarrollo de una propuesta sobre un sistema que permita verificar los horarios de las entradas y las salidas del personal del ayuntamiento mediante el uso de una credencial con código de barras y un lector óptico.

Ésta propuesta está destinada a implementarse en el H. Ayuntamiento constitucional de Ixhuatlancillo o en cualquier institución de carácter público o privado, teniendo como ventaja la sustitución del sistema convencional (registros en formatos impresos en hojas de papel) haciendo de este proceso uno más eficiente.

A continuación de describirá brevemente cada uno de los contenidos presentes en este documento:

INTRODUCCIÓN: Se describirá lo referente a la visualización del objetivo del manual de procedimientos, así como algunos datos relevantes acerca de la empresa en donde se realizó el proyecto de la estadía.

Planteamiento del Problema: Se describirá la situación bajo la cual se realizará el manual de procedimientos.

Objetivos: Lo que se logró al realizar el proyecto.

Estrategias: Son las técnicas, las acciones que se aplicaron al realizar el proyecto.

Metas: lo que se llegó a realizar, el impacto obtenido del proyecto.

Justificación del Proyecto: Se hablará de Todo lo relevante, las razones por las cuales se realizó el proyecto.

¿Cómo y cuándo se realizó? Se describirá ´las fases que se llevaron a cabo a lo largo del proyecto.

Limitaciones y Alcances: Se hablará hasta donde aplica este proyecto, así como el logro obtenido al final del proyecto.

CAPÍTULO 2: Se describirá acerca de la empresa, sus datos generales, antecedentes etc.

CAPÍTULO 3: Se hablará de toda la información teórica utilizada a lo largo de la estadía, datos técnicos, conceptos, etc.

CAPÍTULO 4: En esta parte se hablará de todas las actividades desarrolladas para realizar el proyecto de la estadía.

CAPÍTULO 5: Se hablará de las conclusiones obtenidas al término del proyecto de estadía, los resultados que se obtuvieron así como los posibles trabajos futuros y recomendaciones.

ANEXOS: Aquí se hablará de los datos complementarios y de referencia empleados a lo largo del proyecto.

BIBLIOGRAFÍA: Se mencionarán las fuentes bibliografías que se utilizaron.

INTRODUCCIÓN

En la actualidad existen diversos sistemas que se emplean para formular registros sobre los horarios de las entradas y de las salidas de los trabajadores de forma electrónica por medio del uso de un ordenador y de un software específico, sustituyendo el sistema convencional de llenado a mano de la asistencia de los trabajadores.

Estos sistemas además se cumplir con la función de verificar los horarios de las entradas y de las salidas tienen integradas formas de registro más sofisticadas. Un ejemplo de estos son los checadores con un sistema de reconocimiento facial.

A pesar de la calidad de estos sistemas, estos siguen siendo inaccesibles para muchas empresas, entre ellas las empresas del sector gubernamental, debido a que su adquisición es algo elevada y en diversas ocasiones no llega a cumplir con todos los requerimientos que necesite dicha empresa.
Por lo tanto y analizando el exceso de tiempo al hacer el registro de los horarios de entradas y de salidas del personal del municipio de Ixhuatlancillo Ver., se llegó a la conclusión de hacer un sistema que se encargue de esta tarea y reduzca el tiempo que se desperdicia al hacer el registro convencionalmente.

CAPÍTULO 1
PLANTEAMIENTO DEL PROBLEMA

1.1 Análisis de la situación actual de la empresa

Hacer un cambio de administración en los ayuntamientos de cada municipio del país siempre implica empezar desde cero y esto genera bastantes problemas y gastos repentinos. Uno de estos problemas es la falta de control con respecto a los horarios de trabajo del personal en general. Esto determina si la administración tiene un éxito total o no, debido que este objetivo depende en su totalidad de sus trabajadores y del desempeño que tengan. Los principales factores que intervienen con respecto a los trabajadores son:

-Las horas reales que labora cada trabajador.

-La actitud y la aptitud que cada trabajador con respecto al rol que tiene asignado.

-La manera en que el ayuntamiento administra sus recursos.

Esta propuesta pretende enfocarse en las horas reales que labora cada trabajador. Es de suma importancia que se tenga un control y principalmente un conocimiento de los horarios de las entradas y de las salidas del personal en general. Esto es muy útil cuando se hace un corte y se asignan los salarios correspondientes para cada empleado.

Para ello se pretende hacer un sistema para controlar los horarios de las entradas y las salidas del personal que labora en el ayuntamiento del municipio; con la ayuda de un software y un hardware que sean compatibles con este objetivo y sobre todo que sean eficientes y seguros. Se buscaran los puntos a los cuales se deberán atacar, como son tiempos, recursos, materiales y herramientas disponibles para la realización del mismo; así mismo, como un objetivo extra se va a apoyar en las actividades de mantenimiento asignadas al departamento de alumbrado público.

1.2 Objetivos

Objetivo General:

Elaborar la propuesta de un verificador de entradas y salidas de personal por medio de un software y un lector óptico para generar un mejor control de los horarios de las entradas y salidas del personal.

Objetivos Específicos:

1. Realizar una investigación acerca del software disponible para el desarrollo de un programa que permita crear una base de datos de las entradas y salidas del personal del ayuntamiento.
2. Hacer el registro de cada uno de las personas que laboran en el ayuntamiento.
3. Realizar la cotización correspondiente de cada uno de los materiales que se deban de utilizar para realizar la fase de desarrollo de la propuesta.
4. Verificar el programa ya realizado y hacer las retroalimentaciones correspondientes.
5. Brindar apoyo a las actividades de mantenimiento asignadas al departamento de alumbrado público.

1.3 Justificación del Proyecto

Para que la administración de un municipio tenga éxito y sea eficiente sus colaboradores deben de tener un buen desempeño. Como ya se dijo, los principales factores que influyen en éste desempeño son las horas reales que labora cada trabajador, la actitud y la aptitud que cada trabajador con respecto al rol que tiene asignado y la manera en que el ayuntamiento administra sus recursos.

Por lo tanto, nos vamos a enfocar en las horas en las horas reales que se labora. Es de suma importancia que cada administración de cada municipio tenga un conocimiento, así como un control específico de los tiempos reales en que labora cada uno de los trabajadores.

Es por ello que se presenta la necesidad de buscar o en dado caso en crear un sistema el cual sea capaz de cumplir con las necesidades antes mencionadas y así hacer un control más fácil y eficiente para el personal encargado de supervisar lo referente a los horarios de las entradas y salidas de los trabajadores.

De manera que este proyecto propone la creación de un sistema que cumpla la necesidad de un mejor control de los horarios de las entradas y de las salidas del personal empleando herramientas tecnológicas actuales teniendo dos opciones aplicables a estas necesidades.

1.4 Limitaciones y Alcances

Se refiere a acotar el proyecto, es decir; describir hasta donde aplica el proyecto, y que se pretende obtener al final. Por ejemplo, en el caso de proyectos muy largos

de la empresa, cada proyecto individual de estadía puede atender una sola fase del mismo, por lo que se aclarará también en las conclusiones.

En algunos casos, como corporativos, el proyecto puede ser solo aplicable a una de las empresas o plantas del corporativo.

INSTRUCCIONES:
Letra Arial 12, justificado.
Espaciado de 1.5
Por cada punto y aparte dejar dos espacios.
Mínimo 3 cuartillas

CAPÍTULO 2
DATOS GENERALES DE LA EMPRESA

2.1 Datos generales de la empresa.

Giro de la empresa: Gubernamental.
RFC: MIV941116K82.
Dirección: Av. Independencia No. 85 Colonia Centro, Ixhuatlancillo Ver.
Nº Teléfono: +52 272 72 1 35 70
Fax: ----------

Email: ixhuatlancillover_tesoreria@hotmail.com

El H. Ayuntamiento del Municipio de Ixhuatlancillo Ver. Se encuentra en el municipio de Ixhuatlancillo Veracruz. Es el encargado de administrar los recursos existentes, así como de los ingresos y egresos de los pagos de los servicios proporcionados por éste: tramites sobre la adquisición se servicios como limpia publica, alumbrado público, agua potable, drenaje, deslindes etc.

Toponimia

Es diminutivo de Ixhuatlán, que significa en náhuatl "lugar de las hojas verdes de maíz".

Historia

Desde 1657 los naturales del pueblo de Santa María Asunción Ixhuatlán, pedían amparo de sus tierras con todo el llano de Tlatlauqui, Palan y Tepostlán, de acuerdo con sus derechos antes de la congregación, contra el conde del valle de Orizaba, quejándose de agravios, vejaciones y quema de casas. El virrey accedía al amparo y lo encomendaba a la justicia de Orizaba. El conflicto se prolongó hasta 1741 y 1743. Ya para 1831 se le llamaba al pueblo Santa María Asunción Ixhuatlancillo.

Localización

Se encuentra en la zona central del estado, en las coordenadas 18º 54´ de latitud Norte y 97º 09´ de longitud oeste, a una altura de 1,330 metros sobre el nivel del mar. Limita al norte con la Perla, al este con Mariano Escobedo, al sur con Orizaba, Río Blanco y Nogales, al oeste con Maltrata. Su distancia aproximada al sur de la capital del estado por carretera es de 165 Km.

Extensión

Tiene una superficie de 52.56 Km2, representando el 0.07% del total del estado.

Organización

CAPÍTULO 3
MARCO REFERENCIAL

3.1 Marco de Antecedentes

3.2 Marco teórico

3.2.1 Dev C++

La historia del lenguaje C++

Imagen 1. Bjarne Stroustrup predecesor de la innovación del lenguaje C

C++ es un lenguaje de programación que apareció para mediados de los ochentas y su objetivo fue ampliar el lenguaje C e introducir los nuevos fundamentos de la programación de su día, como ser los objetos y las clases. El lenguaje C++ fue desarrollado por el matemático y doctor de las ciencias de la computación Bjarne Stroustrup.

La historia del lenguaje C++

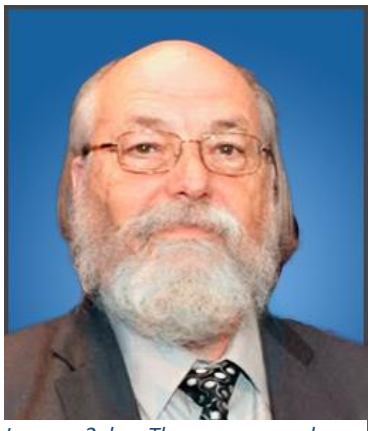


Imagen 2. ken Thompson, creador del lenguaje B en 1970

La historia del lenguaje C++ está estrechamente ligada al desarrollo de los Sistemas Operativos y en concreto al Sistema Operativo UNIX. Cuando **Stroustrup** comenzó a trabajar en AT&T Bell Labs, le asignaron analizar el kernel de **UNIX** y en su investigación se encontró que el lenguaje C le imponía muchas limitaciones a su investigación y como Stroustrup ya contaba con una buena experiencia con Simula, que fue el primer lenguaje en introducir el concepto de clases y objetos, aprovecho

su experiencia para unir la fortaleza de C con la programación orientada a objetos de Simula y fue así como nació el lenguaje de C++.

El lenguaje C++ llego a ser una ampliación y puesta a punto del lenguaje C y el antecesor del lenguaje C fue el Lenguaje B, que fue creado en 1970, por Ken Thompson, con la finalidad de reestructurar al sistema **UNIX.** El Lenguaje B a su vez utilizo como base de su desarrollo a BCLP que fue una simplificación del lenguaje CPL.

Nombre de C++

El nombre de **C++** fue recomendado por Rick Mascitti en el año 1983, y se usó **++**porque es el *operador de incremento* así el nombre de que C++ indica que es un incremento o extensión del **lenguaje C.**

En un principio C++ no contaba con un nombre oficial así que comenzó a recibir sobre-nombres y muchos lo llegaron a conocer como **C con Clases**

ISO C++

En la actualidad se maneja un estándar para el desarrollo del lenguaje C++ denominado ANSI C++ este Estándar tuvo su inicio en 1983 por el *Instituto Nacional Estadounidense de Estándares* y su arduo trabajo termino en 1989 dando como resultado la primera estandarización del lenguaje con el nombre **C++89** que ha ido evolucionando hasta llegar al estándar **C++17.**

Java

En la actualidad los lenguajes que originalmente eran no estructurados han sido modificados para que cumplan con esta característica, tal es el caso de Basic, que actualmente soporta la programación orientada a objetos. Podemos notar cuando un lenguaje de programación es viejo si vemos que no cumple con la programación estructurada. C++ es, también, un lenguaje orientado a objetos, y es el mismo caso

de Java. Al referirnos a lenguaje estructurado debemos pensar en funciones, y también a sentencias de control (if, while, etc.)

Muchos compiladores de C++ están orientados hacia el desarrollo bajo entornos gráficos como Windows 98. Este sistema operativo está escrito casi completamente en C, incluso cuando la compañía Microsoft creó el compilador Visual C++. “Pero deberíamos preguntarnos si esta aparente anomalía refleja una mentalidad inmadura de C++ entre los diseñadores del este sistema y el deseo de la compañía de influir en el código de sistema operativo existente o una relativa idoneidad de los dos lenguajes para escribir sistemas operativos”2. En lo personal, pensaría en otra razón popular entre usuarios de Linux. C++ es un supe conjunto de C, cualquier compilador de C++ debe ser capaz de compilar un programa en C. De hecho la mayoría admite tanto código en C como en C++ en un archivo. Por esto, la mayoría de desarrolladores compilan con C++ su código escrito en C, incluso hay quienes, siendo código en C ponen la extensión CPP (extensión de los archivos de código C++) y lo compilan con C++ (hasta hace unos días yo hacía esto), lo cual no es recomendable por norma al programar. Cuando se compila código C en un compilador C++ este debe cumplir con los estándares definidos en 1989, cualquier palabra definida en el estándar de 1999 no funcionará.

La evolución de C++ continúa, y la diversidad de compiladores contribuye a ello.

Lenguajes asociados

C++ Builder Creado por una empresa de nombre Inprise, sobradamente conocida por la calidad de sus herramientas de desarrollo, que llevan la firma Borland, entre las que están Borland C++, Intrabuilder, JBuilder, Delphi, C++ Builder. C++ Builder surgió de la fusión de dos tecnologías: Borland C++ que soporta el lenguaje C++ estándar con todas sus novedades, y el entorno RAD de Delphi. Visual C++ En el año de 1992, la compañía Microsoft introduce C/C++ 7.0 y la biblioteca de clases MFC, los cuales tenían la finalidad de que el desarrollo de aplicaciones para

Windows, escritas en C, fuera más fácil. Sin embargo, no resultó como esperaban, así que un año más tarde fue creado Visual C++ 1.0, que parecía más amigable a los desarrolladores, porque tenía una versión mejorada de las clases MFC. Con Visual C++ se introdujo una tecnología de desarrollo a base de asistentes. En general es un entorno de desarrollo diseñado para crear aplicaciones gráficas orientadas a objetos. Pero si cree que se trata sólo de arrastrar componentes hacia un “formulario”, como lo puede hacer en Visual Basic, está muy equivocado. La programación en él es más compleja que eso.

Visual **C#**

Este lenguaje es una evolución de C y C++. Creado por Microsoft y presentado como Visual C# en el conjunto de Visual Studio .NET. Está diseñado para crear una amplia gama de aplicaciones empresariales. La biblioteca para programar en este lenguaje es .NET Framework. El lenguaje es simple, moderno, y está orientado a objetos. El código de C# se compila como código administrado, lo que significa que se beneficia de los servicios de Common Language Runtime. Estos servicios incluyen la interoperabilidad entre lenguajes, la recolección de elementos no utilizados, mayor seguridad y mejor compatibilidad entre las versiones.Java El lenguaje de programación Java se había creado como una plataforma para desarrollo de sistemas de información para Internet y para aparatos electrodomésticos. Fue creado con base en C++. Poco tiempo después de su liberación (en 1995), se empiezan a ver las capacidades de este lenguaje. Pronto deja de ser un lenguaje que sólo se usaba en Internet, y empiezan a crearse programas completos con él.

El lenguaje es orientado a objetos y multiplataforma, esto quiere decir que el código se puede transportar a diferentes sistemas operativos y ejecutarse en ellos por medio de la máquina virtual de Java.

Al igual que con C++ existen varios compiladores para este lenguaje como lo son JBuilder y Visual J++ (ahora evolucionado a J#).

Estructura básica de un programa en C

#include <iostream.h> }declaración de librerías int main(void){ }función main cout<<"hola mundo"<<endl; }secuencia de instrucciones
return 0; }valor de retorno de la función
} }llaves de cierre de la función

Tipos de datos en c++

En la sección anterior vimos la forma general de un programa, un programa sumamente sencillo. Ahora veamos un programa muy parecido al anterior:

```
#include <iostream.h>
int main( ){ int
variable;
variable=5;
cout<<variable;
return 0;
}
```

Notemos en esta ocasión sólo la parte: int variable; A esta sección se le denomina declaración. Se trata de la declaración de una variable de nombre "variable". Una variable es una posición de memoria con nombre que se usa para mantener un valor que puede ser modificado por el programa3. Las variables son declaradas, usadas y liberadas. Una declaración se encuentra ligada a un tipo, a un nombre y a un valor. Básicamente, la declaración de una variable presenta el siguiente aspecto: tipo nombre [=valor]; Los corchetes significan que esa parte es opcional. Por

ejemplo, la declaración: int mi_variable=5; declara una variable tipo entero de nombre “mi_variable” y le asigna el valor “5”.

C++ es sensible a mayúsculas y minúsculas, así que si el nombre de nuestra variable empieza con una letra en mayúsculas, debemos de asegurarnos que durante el resto del código nos refiramos a ella exactamente como la escribimos. Los nombres de las variables no pueden usar signos de puntuación, sólo caracteres “A-Z”, “a-z”, “_“, “0- 9”, aunque ningún nombre debe comenzar con un número (0-9). Además no se deben de repetir nombres de variables en el mismo contexto. Además de las restricciones anteriores, existe otra, y esta tiene que ver con las palabras reservadas del lenguaje, que no son muchas a comparación de otros lenguajes como Basic.

Las variables se pueden declarar en tres sitios básicos: dentro de las funciones (ya sea la función main u otras creadas por el programador), estas variables son llamadas locales; en la definición de parámetros de una función, como se verá más adelante; y fuera de todas las funciones, variables globales.

Ejemplo:

```
#include <iostream.h> int variable_global=10; int main(){
int variable_local=20; cout<<"\nprograma que muestra los
usos de variables "\
"globales y locales\n"<<endl;
cout<<"la variable global tiene asignado un: "\
<<variable_global<<endl;
cout<<"\nla variable local tiene asignado un: "\
<<variable_local<<endl;
return 0;
}
```

Una variable global puede ser modificada en cualquier parte del programa, mientras que una variable local sólo puede ser modificada y utilizada dentro de la función en

la que se ha declarado. Por supuesto, antes de utilizar una variable y hacer operaciones con ella, hay que declararla.

Por lo general, siempre se trata de utilizar lo menos posible la declaración de variables globales. El siguiente ejemplo muestra que se pueden declarar variables en cualquier parte del programa, siempre y cuando se declaren antes de usarlas.

```
#include <iostream.h>
int main( ){ int
variable1=10;
cout<<"la variable 1 local tiene almacenado un: "\
<<variable1<<endl;
variable1=50;                int
variable2=variable1+30;
cout<<"\nla variable 2 almacena un: "\
<<variable2<<endl;
return 0;
}
```

En un programa puede que necesitemos declarar un dato y asignarle un nombre, pero que éste no pueda ser modificado. En este caso debemos declarar una constante. Por ejemplo, el siguiente programa calcula el área de un círculo.

```
#include <iostream.h>
int main( ){ const float
pi=3.141592;
int radio=5; float area; area=pi*radio*radio;
cout<<"el area del circulo es:
"<<area<<endl; return 0;
}
```

Declaramos una constante del tipo de datos float , le damos el nombre “pi” y le asignamos el valor 3.141592. Este valor jamás podrá ser modificado en ninguna parte del programa La declaración de una constante presenta el siguiente aspecto:

const tipo nombre = valor;

Tipos de datos

Los prototipos de variables utilizados en los programas de ejemplo, hasta el momento, han sido en su mayoría de tipo entero (int), pero es ilógico pensar que éste sea el único que se llegue a utilizar. Además del tipo entero existen otros. Los tipos de datos atómicos son los tipos de datos más sencillos a partir de los cuales se pueden construir otros más complejos. La siguiente tabla ilustra estos tipos con sus intervalos de valores posibles y el número de bytes que ocupan.

Tipo Tamaño en bytes Intervalo
short 2 -32768 a 32767 unsigned
short 2 0 a 65535
long 4 -2147483648 a 2147483647
unsigned long
4 0 a 4294967295 int
dependiendo del
compilador utilizado
podría ser 2 o 4 -32768
a 32767
unsigned
int
2 o 4 0 a 65535
float 4 1.17549535e-38 a 3.402823466e+38 con 8 cifras
decimales

double 8 2.2250738585072014e-308 a 1.7976931348623158e+308 con 16 cifras decimales

long double 10 char 1 -128 a 127 unsigned char 1 0 a 255 bool 1 Valor lógico o booleano que puede ser trae (cierto) o false (falso).

En la biblioteca <limits.h> encontrará constantes definidas para los intervalos que pueden almacenar algunos tipos de variables. Compile y ejecute el siguiente programa, después consulte la biblioteca <limits.h> para ver más constantes definidas.

```
#include <limits.h> #include
<iostream.h>
int main( ){
cout<<"PROGRAMA QUE MUESTRA LOS VALORES MAXIMOS Y "\
"MINIMOS \n DE ALGUNOS DE LOS TIPOS DE DATOS "\
"ATOMICOS"<<endl;
cout<<"\n int maximo: "<<INT_MAX<<" int minimo: "\
<<INT_MIN<<endl; cout<<"\n char maximo:
"<<CHAR_MAX<<" char minimo: "\ <<CHAR_MIN<<"
tamaño en bits: "<<CHAR_BIT<<endl; cout<<"\n long
maximo: "<<LONG_MAX<<" long minimo: "\
<<LONG_MIN<<endl;
cout<<"\n short maximo: "<<SHRT_MAX<<" short minimo: "\
<<SHRT_MIN<<endl;
return 0;
}
```

Por medio de los ejemplos dados hasta el momento nos podemos dar una idea de la forma en que se asignan valores a los tipos de datos numéricos. Tan sólo falta

mostrar cómo se pueden asignar valores en formato octal y hexadecimal. Las siguientes instrucciones lo aclararán.

int variable1=022; int
variable2=0x12;

Ambas variables están almacenando el número 18, la primera en octal y la otra en hexadecimal. También se pueden asignar números exponenciales.

float variable3=2e6;

Pero no sabemos nada sobre los de tipo char.

Para declarar y asignar un carácter a una variable se hace de la siguiente forma.

char nombre = '[carácter]';

Y si es una constante sólo se agrega la palabra const al principio de la declaración. Note que el carácter asignado está encerrado entre comillas simples. Así, la siguiente declaración asigna la letra "A" mayúscula a una constante de nombre "inicial". onst char inicial='A';

El valor (carácter) que se almacena en una variable o constante de tipo char, es el valor entero del carácter en el juego de caracteres ASCII. Entonces a la constante "inicial" se le asignó un valor de 65.

Para representar caracteres de control es necesario usar secuencias de escape. Caracteres de control

Significado

\n salto de línea

\t tabulador horizontal

\v tabulador vertical

\a alerta sonora

\0 carácter nulo \b mueve
el cursor hacia atrás

Cualquier carácter se puede representar con la “\” seguida del propio carácter o del código octal o hexadecimal. Por ejemplo las comillas dobles serán ‘\”’ o ‘\042’ o ‘\x22’.

Operaciones básicas

En ejemplos anteriores ya hicimos uso de algunos operadores, “+” y “*”, suma y multiplicación respectivamente.

Operador
Significado
+ adición
- sustracción
* multiplicación
/ división
% resto de la división entera

El siguiente programa ilustrará mejor.

```
#include <iostream.h>
int main( ){ int a=5,
b=10, c=20, r;

r=a+b; a=c%r; //aquí “a” valdrá 5(resto de 20/15) c=ba;
a=a*2; cout<<"a="<<a<<" b="<<b<<" c="<<c<<"
r="<<r<<endl; cout<<"la suma de a y b es: "<<a+b<<endl;
return 0;
```

}

Estos operadores no son exclusivos de los tipos de datos numéricos. Un dato tipo char también es modificable mediante operadores. Recuerde que lo que se guarda es el código ASCII.

```
#include <iostream.h> int main( ){ char car1='a',
car2; car2=car1+15; cout<<"car2="<<car2<<endl;
car1=car210; cout<<"car1="<<car1<<endl;
cout<<"la suma de los dos
es:"<<car1+car2<<endl; car1=car1car2;
cout<<"car1="<<car1<<endl;
return 0;
```

Entrada y salida básica

Entrada y salida

En los programas hechos hasta el momento, hemos utilizado la instrucción cout<< para mandar mensajes a la pantalla. La mayoría de los programas en C++ incluyen el archivo de encabezado <iostream.h>, el cual contiene la información básica requerida para todas las operaciones de entrada y salida (E/S) de flujo. Cuando usamos la instrucción:

cout<<”Mensaje a la pantalla”<<endl; Estamos enviando una cadena de caracteres (“Mensaje a la pantalla”) al dispositivo de salida estándar (la pantalla).

Luego, el manipulador de flujo endl da el efecto de la secuencia de escape ‘\n’. Pruebe el siguiente programa:

```
#include <iostream.h>
int main(){
```

```
cout<<"cadena de caracteres"<<endl;
cout<<2+2<<endl; //imprime un entero
cout<<9/2<<endl; //imprime un flotante
cout<<(int)(3.141592+2)<<endl; //imprime un entero
31  return
0;
}
```

La instrucción cout<< puede imprimir tanto números enteros como flotantes sino necesidad de decirle específicamente el tipo de datos del que se trata, pero, por supuesto notemos que al enviarle una cadena de caracteres esta debe de estar entre comillas. Ya en ejemplos anteriores vimos como se mandaba a la pantalla el valor de una variable, así que no hace falta más ilustración al respecto. La interacción con el usuario es algo muy importante en la programación, imaginemos que en este preciso momento y con los conocimientos que tenemos hasta ahora, necesitamos hacer un programa que calcule la distancia a la que caerá un proyectil lanzado a determinada velocidad y ángulo, o simplemente un programa que calcule las raíces de una ecuación cuadrática. Sería muy molesto estar cambiando los valores de las variables directamente en el código para cada caso que queramos calcular. Por eso debemos ver cuanto antes la forma de leer datos desde el teclado.

La principal función para leer desde el teclado es cin>>, pero es mejor ver un ejemplo para tener la idea clara. #include <iostream.h>

```
int main(){ int numero; char car; float
otroNum;  cout<<"escribe  un
numero:"<<endl; cin>>numero;
cout<<"\nel numero que tecleaste es: "<<numero<<endl;
cout<<"dame una letra"<<endl; cin>>car;
cout<<"\ntecleaste:  "<<car<<endl;  cout<<"escribe
un numero flotante"<<endl;
```

```
cin>>otroNum; cout<<"\neste es:
"<<otroNum;
return 0;
}
```

En resumen, cin es el flujo de entrada asociado al teclado, cout es el flujo de salida estándar asociado a la pantalla, y existe otro, que aunque a lo largo de este trabajo casi no lo utilicemos, es necesario nombrarlo, cerr, que es el flujo de error estándar asociado a la pantalla. Los operadores << y >> son operadores de inserción y extracción de flujo respectivamente, y no deben confundirse con los de desplazamiento de bits. Estos operadores son muy eficaces porque no es necesario especificar formatos para presentación o lectura, ellos los presentan en función al tipo de datos de la variable. Aunque en ocasiones podría necesitar de nuestra ayuda para obtener los resultados específicos que queremos, y para eso están los modificadores de formato.

Modificadores de formato

En el primer ejemplo del presente capítulo quizá no entendió del todo lo qué hacía la instrucción: cout<<(int)(3.141592+2)<<endl;

Si probó el código y lo ejecutó seguramente esperaba que apareciese en la pantalla 5.141592, esto habría pasado si no hubiésemos hecho una conversión de tipo. La inclusión de “(int)” dentro de la instrucción provocó que el resultado de la operación, que debía ser 5.141592, se transformara a un entero y por resultado perdiera sus valores decimales.

A esto se le llama hacer un cast, y es muy común hacerlo cuando no queremos que se pierdan determinadas propiedades en los resultados de las operaciones. Por

ejemplo, las siguientes líneas de código, que son equivalentes tienen por efecto convertir el tipo de la variable de origen al de la variable destino:

int B; int B; double A= (double) B; double A = double(B); Otra de las herramientas para la especificación de formatos son los manipuladores de flujos, así como el manipulador de flujo “endl” da una secuencia de escape, los manipuladores dec, oct y hex, hacen que la variable a la que le anteceden sea presentada en formato decimal, octal o hexadecimal respectivamente.

```
#include <iostream.h>
int main(){ int
numero=25;
int leido;
cout<<"numero es en octal:
"<<oct<<numero<<endl; cout<<"en hexadecimal:
"<<hex<<numero<<endl; cout<<"en decimal:
"<<dec<<numero<<endl; cout<<"ahora teclea un
numero"<<endl; cin>>hex>>leido; cout<<"el leido
vale: "<<hex<<leido<<endl; cout<<"y en decimal:
"<<dec<<leido<<endl; cout<<"y en octal:
"<<oct<<leido<<endl; return 0;
}
```

Estos manipuladores de flujo no son exclusivos de la salida, también funcionan con la entrada de datos, por defecto, la entrada desde el teclado se lee en decimal y también la presentación de datos se hace en decimal, así que si se sabe que el usuario dará a la máquina un dato en hexadecimal, sería buena idea anticiparlo y poner el formato en que será leído.

En cuanto a la precisión, al imprimir en pantalla un número de tipo flotante, no importa cuántas cifras significativas tenga (mientras esté dentro de los límites de almacenamiento de flotantes), sólo aparecerán 6 números después del punto, y la última cifra será redondeada en caso de pasar los 6 dígitos. Es decir, que si queremos mandar a la pantalla el número 1.23456789, en su lugar aparecerá 1.234568.

Cuando se quieren resultados más exactos, nos es muy útil la librería iomanip.h que contiene el manipulador setprecision() con el que podemos determinar la precisión con la que queremos que aparezcan esos datos en la pantalla. Existe también la función miembro pecision(), que puede resultar un poco más cómoda en algunos casos porque el efecto de precisión funciona para todas las instrucciones cout que le sigan. A ambas se le pasa como parámetro el número de cifras significativas deseadas. Veamos un ejemplo.

```
#include <iostream.h> #include
<iomanip.h> int main(){
cout<<1.23456789<<endl;
cout.precision(4);
cout<<1.23456789<<endl;
cout<<setprecision(5)<<1.23456789<<endl
;
cout.precision(7);
cout<<1.23456789<<endl;
cout<<setprecision(9)<<1.23456789<<endl
; return 0;
}
```

En ocasiones es útil dar una presentación a nuestros datos en la pantalla, tan sólo el dejar unos cuantos espacios de margen en la pantalla puede ser un poco tedioso

para quienes no están enterados de las siguientes funciones: setw que se encarga de hacer aparecer el texto alineado determinados espacios a la derecha, si el numero de espacios solicitados para su alineación es menor que el numero de caracteres que se imprimirán, no tendrá un efecto. setfill se ocupa del carácter de relleno, el carácter por defecto es el espacio en blanco, pero nosotros podemos especificar el que queramos.

Estructuras de control

En nuestra vida cotidiana, todos tenemos una lógica a seguir, continuamente tomamos decisiones, y estas decisiones repercuten en nuestra acción siguiente. Por ejemplo, supongamos el caso de un estudiante de nivel preparatoria que cursa el sexto semestre, él está pensando en presentar el examen para la universidad, sin embargo, sus calificaciones no le han dado mucho aliento últimamente, y está en riesgo de tener que repetir ese semestre, si ocurre eso, el resultado que tenga en el examen no importará. Lo que valla a pasar marcará el camino a seguir en su vida.

Analicemos de una forma general este caso:
curso el sexto semestre presento el examen de admisión si paso el examen y además paso el semestre estaré en la universidad si paso el semestre pero no paso el examen estaré trabajando si no paso el semestre curso el sexto semestre (es decir, regresa al principio). Estas son las opciones que él se plantea, aunque algunos pensarán en otras más.

Puede estudiar, trabajar o repetir todos los pasos anteriores. En la programación también se tienen que tomar decisiones que influirán en el comportamiento del programa, y también se pueden repetir series de pasos hasta obtener un resultado. En el presente capítulo aprenderemos eso. If Se trata de una estructura de selección. Es decir que si se cumple el condicional se ejecutarán varias instrucciones más. En el ejemplo anterior, observamos que:

-si paso el examen y además paso el semestre estaré en la universidad.

“si” podemos interpretarlo como el if de código c++ “paso el examen y además paso el semestre” es la condición “estaré en la universidad” es la acción a ejecutar.

Escribamos el código:

```
#include <iostream.h>
int main(){
char examen[2],semestre[2]; cout<<"Contesta
si o no a las preguntas"<<endl;
cout<<"Pasaste el examen?"<<endl;
cin.get(examen,2);
cin.ignore(20,'\n');
cout<<"Pasaste el semestre?"<<endl;
cin.get(semestre,2); cin.ignore(20,'\n'); //ignora el enter
if((examen[0]=='s')&&(semestre[0]=='s')){
cout<<"estas en la universidad"<<endl;
}
cout<<"fin del programa"<<endl;
cin.get(); //espera a que oprimas enter
return 0;
}
```

Analicemos lo que nos importa, la sentencia if. La condición a cumplir es: (examen[0]= =’s’)&&(semestre[0]= =’s’), en realidad se trata de dos condiciones que se deben de cumplir: que la variable examen tenga el carácter ‘s’ y que la variable semestre tenga el carácter ‘s’ (vease la tabla 7 operadores relacionales y de igualdad). Si la primera condición se cumple, entonces comprueba la segunda condición (gracias a la conjunción lógica &&).

Un condicional regresa 1 en caso de ser verdadero y un 0 en caso de ser falso, esto dos valores, en c++ están definidos como tipos de datos booleanos, true y false respectivamente, así que si el condicional resulta true (verdadero), se evalúa la siguiente instrucción dentro del bloque if, en caso contrario no hace nada.

En el ejemplo anterior el bloque de instrucciones a ejecutar en caso de ser verdadera la condición se encuentra dentro de llaves, pero no es necesario cuando sólo se trata de una sola línea de código.

Cuando queremos, además controlar las opciones en caso de que la condición resulte falsa, tenemos que incluir aquellas instrucciones en un bloque else. Por ejemplo, en el código anterior podemos añadir un else a la sentencia if, de manera que nos quede de la siguiente forma:

```
if((examen[0]=='s')&&(semestre[0]=='s')){
cout<<"estas en la universidad"<<endl;
}
else{
cout<<"no estas en la universidad"<<endl;
}
```

Igualmente el bloque de instrucciones a ejecutar se estructura dentro de llaves, aunque tampoco es necesario cuando se trata de una sola instrucción.

En ocasiones necesitaremos construir bloques de instrucciones if y else más complejos, es decir anidamientos. El ejemplo del estudiante también nos ayudará a visualizar este concepto. si paso el examen y además paso el semestre estaré en la universidad si paso el semestre pero no paso el examen estaré trabajando si no

paso el semestre curso el sexto semestre Construyendo mejor esta serie de decisiones, podemos acomodarla de la siguiente forma, al estilo seudocódigo.

if paso el examen y además paso el semestre estaré en la universidad else if paso el semestre pero no paso el examen estaré trabajando else curso el sexto semestre De esta manera controlamos las posibles respuestas que pudiese dar. En caso de que pase el examen y el semestre estará en la universidad, si pasa el semestre pero no pasa el examen estará trabajando, y en cualquier otro caso, cursará el semestre.

Nuestro código ya terminado quedará de la siguiente manera:

```
#include <iostream.h>
int main(){
char examen[2],semestre[2]; cout<<"Contesta
si o no a las preguntas"<<endl;
cout<<"Pasaste el examen?"<<endl;
cin.get(examen,2);
cin.ignore(20,'\n'); cout<<"Pasaste el
semestre?"<<endl;
cin.get(semestre,2); cin.ignore(20,'\n');
//ignora el enter
if((examen[0]=='s')&&(semestre[0]=='s'
)) cout<<"estas en la
universidad"<<endl; else
if((examen[0]=='n')&&(semestre[0]=='s'
)) cout<<"estaras trabajando"<<endl;
else cout<<"cursa el sexto
semestre"<<endl; cout<<"fin del
programa"<<endl; cin.get(); //espera a
que oprimas enter return 0;
```

```
}
```

En esta ocasión se eliminaron las llaves que no eran necesarias.

Este se trata de un operador del tipo condicional al estilo if/else. Se utiliza para condicionales cortos (de una sola línea).

Por ejemplo:

```
#include <iostream.h>
int main(){ int calificacion; cout<<"Cual fue tu calificacion del semestre?"<<endl; cin>>calificacion; cout<<(calificacion>=6 ? "pasaste" : "reprobaste"); cin.ignore(); cin.get();
return 0;
}
```

La forma de este operador es la siguiente:

condición ? acción a ejecutar en caso verdadero : acción a ejecutar en caso falso ;

La condición es: calificación>=6. En caso de ser verdadera se enviará “pasaste” como argumento a la instrucción cout, en caso contrario se enviará reprobaste.

switch …

Muchas veces nos metemos en aprietos cuando necesitamos tener el control sobre muchas opciones que pudiese tomar el usuario, porque resulta muy complicado pensar en varios if/else anidados, para esos casos tenemos otra herramienta muy cómoda, la estructura de selección múltiple switch. La forma general es:

```
switch (parámetro a evaluar o comparar){ case a
: //cuando el parámetro tiene un valor a Acciones
a ejecutar;
```

```
case b: //cuando el parámetro tiene un valor b Acciones
a ejecutar
.
.
caso por default;
}
#include <iostream.h>
int main(){
int opcion; cout<<"Menu de
opciones"<<endl;
cout<<"1.- Opcion 1"<<endl;
cout<<"2.- Opcion 2"<<endl;
cout<<"3.- Opcion 3"<<endl;
cout<<"elige una opcion"<<endl;
cin>>opcion; switch(opcion){ case
1: cout<<"ejecucion 1"<<endl;
break; case 2: cout<<"ejecucion
2"<<endl; break; case 3:
cout<<"ejecucion 3"<<endl; break;
default:
cout<<"es una opcion no valida"<<endl; break;
}
cout<<"presiona enter para salir"<<endl; cin.ignore();
cin.get();
return 0;
}
```

Notemos que “opcion” es una variable de tipo entero, asi que case 1 , se refiere a que la variable “opcion” tiene un valor de 1. Si se tratase de una variable de tipo carácter, entonces debería de escribirse case ‘1’.

Encontramos una nueva instrucción, break. Esta se encarga de alterar el flujo de control. Cuando se encuentra un break, inmediatamente se salta el resto de la estructura. No es exclusiva de la estructura switch, también se puede colocar en bloques if, for, while o do while (que se verán a continuación). Sin embargo es mejor no depender demasiado del break.

While

La sentencia de control while se encarga de repetir un bloque de código mientras se cumpla una condición. El bloque de código se debe de encontrar entre llaves, excepto si es una sola línea. La forma general de esta instrucción es la siguiente:

while (condición a cumplir) { acciones a ejecutar;

}

Un ejemplo de éste lo podemos ver en el cálculo del promedio de 10 números, pensemos que son las calificaciones del semestre. Mirando rápidamente la forma de estructurar nuestro programa (algo que se debe de hacer siempre, y antes de sentarse frente a nuestro ordenador), podemos pensar en la siguiente solución:

Declarar variables e inicializarlas

Mientras (no se hagan 10 iteraciones){

Leer una calificación

Sumarla al total

Determinar que se ha hecho una iteración

}

calcular el promedio mostrar el promedio

El programa queda como sigue: #include <iostream.h>

int main(){ int calificacion, suma=0, iteracion=1;

float promedio; while(iteracion<=10){

```
cout<<"teclea tu calificacion "<<iteracion<<endl;
cin>>calificacion; suma+=calificacion;
++iteracion;
}
promedio=(float)suma/(iteracion1); cout<<"el
promedio de calificaciones es: "
<<promedio<<endl;
cin.ignore(); cin.get();
return 0;
}
```

El programa es sencillo, quizá una parte que pudiese causar dudas es el cálculo del promedio. Lo que sucede es que se realiza un cast para convertir temporalmente la variable suma de int a float, así se hará la división y se tendrá un resultado decimal cuando sea necesario. Hasta aquí nuestro programa va bien, pero puede mejorar, podemos tener un mejor control del programa y hacer que el usuario decida cuantas calificaciones va a introducir. Podemos hacer que el usuario simplemente teclee un "-1" cuando acabe de escribir sus calificaciones, así, en lugar de comparar el número de iteraciones, comprobará si la calificación es igual a "-1" para salirse.

Aquí el programa:

```
#include <iostream.h>
int main(){ int calificacion=0, suma=0,
iteracion=0; float promedio=0;
while(calificacion!=-1){
++iteracion; suma+=calificacion; cout<<"teclea tu
califiaccion "<<iteracion<<endl; cin>>calificacion;
}
promedio=(float)suma/(iteracion1); cout<<"el
promedio de calificaciones es: "
```

```
<<promedio<<endl;
cin.ignore(); cin.get();
return 0;
}
```

Podemos ver como cambiando unas instrucciones de lugar y un valor obtenemos un mejor resultado. La variable “iteracion” la inicializamos con un valor de 0, al inicio del bucle le aumentamos una unidad, luego a “suma” le añadimos la calificación que en este caso aún es 0, luego pide y lee la calificación tecleada. Al siguiente ciclo comprueba si se escribió “-1”, si no entonces aumenta el contador “iteracion”, suma la calificación introducida, por ejemplo un 8, y pide la siguiente, el usuario teclea, de nuevo, 8.

Hace de nuevo una comprobación y entra de nuevo al ciclo, aumenta el valor de “iteracion” a hora tiene un 3, “suma” ahora vale 8 que era lo que tenía antes más el 8 recientemente leído, es decir 16, pide la siguiente calificación, y teclea la persona “-1”.

Otra comprobación da como resultado un falso (por que se escribió -1) y por lo tanto no entra al ciclo, entonces calcula el promedio, donde notemos que a “iteracion” se le resta uno para tener el verdadero número de calificaciones introducidas. Por último presenta el resultado.

Hay otras formas de hacer este programa, por ejemplo cuando no esta inicializada la variable de calificaciones o iteraciones, y necesitamos que ese ciclo se ejecute por lo menos una vez. En este caso podemos usar algo muy parecido, se trata de la sentencia do while.

do while

Cuando necesitamos que un ciclo se ejecute por lo menos una vez, es necesaria esta sentencia. La forma general es:

```
do{
Acciones a ejecutar;
}
while(condicion a cumplir);
```

Pongamos como ejemplo el mismo de la estructura switch, el menú de opciones. Necesitamos que por lo menos una vez lea la opción que elige, y en caso de que elija una no disponible, en lugar de aparecer el mensaje “es una opción no válida” y termine el programa, espere a que el usuario escoja una de la lista. El programa de ejemplo:

```
#include <iostream.h>
int main(){ int opcion; do{ cout<<"elije una
opcion de la lista"<<endl;
cout<<"1.-opcion
1"<<endl;       cout<<"2.-
opcion            2"<<endl;
cout<<"3.-opcion
3"<<endl; cin>>opcion;
}while((opcion<1)||(opcion>3));   cout<<"esta
opcion si fue valida"<<endl;
cin.ignore(); cin.get();
return 0;
}
```

Nunca debemos olvidar el punto y coma después del while. Usada en combinación con el bloque switch, podemos tener muy buenos resultados y encontrar varias aplicaciones de ellos.

For

El ciclo for es al que más se recurre cuando se requiere realizar operaciones secuenciales, en donde se conoce el número de iteraciones o la condición a comprobar. La forma general de esta sentencia es:

for(asignación inicial ; condición ; instrucción) {
bloque de instrucciones;
}

Si se trata de una sola instrucción a ejecutar no es necesario ponerla entre llaves, incluso podría ponerse dentro del paréntesis el for, separado por una coma.
for(asignación inicial ; condicion ; instrucción1) unica instrucción; ó, si se trata de una instrucción muy corta podría ponerse:

for(asignación inicial ; condicion ; instrucción1 , instrucción 2)
Pensemos en una aplicación. Hagamos un programa que calcule las coordenadas de un proyectil que se mueve en movimiento parabólico. Almenos determinaremos 10 puntos por los que pasa. Si se tienen dudas acerca de las fórmulas recomiendo que consulten un libro sobre física.

Las formulas son las siguientes:
x=ðß Viðàðßsinðïðàðß tðà y=ðß
Viðàðßsinðïðàðß tðàðÄ
1 2 ðß gðàðß
t2ðà
Donde Vi es la velocidad inicial.
ðï es el ángulo con el que es lanzado. t
es el tiempo. g es la gravedad.
La forma en que pensaría hacer este programa es la siguiente:

Declarar e inicializar variables Pedir datos

for (contador=1; hasta que el contador llegue a 10; aumentar contador){

Calcular coordenada x Calcular
coordenada y

Imprimir las coordenadas

}

Termina el programa.

No parece tan complicado ¿verdad? Pues en realidad no lo es tanto. El programa terminado aquí está. #include <iostream.h> #include <math.h> int main (){ int t;

float x=0,y=0,ang,vi, grav=9.81; cout<<"cual es el
angulo de lanzamiento?"<<endl; cin>>ang;
cout<<"cual es la velocidad inicial?"<<endl; cin>>vi;

ang=((ang*3.141592)/180); for(t=0;t<=10;t++){
x=(vi*cos(ang)*t); y=(vi*sin(ang)*t)-(
0.5*grav*t*t); cout<<"t= "<<t<<" x= "<<x<<" y=
"<<y<<endl;

}

cout<<"fin del programa"<<endl;

cin.ignore(); cin.get();

return (0);

}

El ciclo for comienza inicializando la variable del tiempo en 0. La condición que se tiene que cumplir para entrar en el ciclo es que t sea menor o igual a 10. Cuando entra en el ciclo calcula las coordenadas x y y. Hacemos uso de las funciones cos y sin que se encuentran dentro de la librería math.h. Dichas funciones calculan el coseno y el seno de un ángulo determinado en radianes, es por eso que la variable “ang” es transformada a radianes. Luego de haber calculado las coordenadas, las imprime en pantalla. Y finalmente ejecuta la última instrucción del for aumentar una unidad a t (t++).

En el momento en el que la variable t sea mayor que 10, no entrará en el ciclo y llegará el fin del programa.

Funciones

Cuando tratamos de resolver un problema, resulta muy útil utilizar la filosofía de “divide y vencerás”. Esta estrategia consiste en dividir nuestro problema en otros más sencillos Cuando realizamos un programa, por ejemplo, en el que se repetirán varias instrucciones pero con distintos valores que definan los resultados, podemos construir el programa a base de funciones. Una función es un bloque de instrucciones a las que se les asigna un nombre. Entonces, cada que necesitemos que se ejecuten esa serie de instrucciones, haremos una invocación a la función.

Ya hemos hecho uso de algunas funciones, en el capítulo anterior usamos cos y sin, que están presentes en la librería math.h. Así no nos preocupamos por todo lo que haga esa función para devolver un resultado, lo único que nos preocupó fue que datos teníamos que mandar y los que íbamos a recibir.

Prototipos

El prototipo de una función se refiere a la información contenida en la declaración de una función. Una función debe de estar definida o al menos declarada antes de hacer uso de ella. Cuando se declara una función debe de especificarse el tipo de dato que va a devolver, el nombre de la función, y los parámetros. La siguiente declaración:

```
int suma(int a, int b);
```

Especifica una función que devuelve un tipo de dato entero, tiene por nombre suma, y al invocarla, recibe 2 parámetros de tipo entero.

Esta declaración debe de escribirse antes de la función main, y su definición puede escribirse después de ésta.

Por ejemplo:

```
#include <iostream.h> int suma(int x, int y); int
main(){ int dato1,dato2; cout<<"calculare la suma
de 2 numeros"<<endl; cout<<"escribe el dato 1 y
luego el dos"<<endl; cin>>dato1; cin>>dato2;
cout<<"la suma es: "<<suma(dato1,dato2)<<endl;
cin.ignore(); cin.get();
return 0;
}
int suma(int a, int b){ return
a+b;
}
```

Observamos que primeramente declaramos la función suma para usarla en el bloque main, y al final del código la definimos. Podíamos haber escrito la función completa antes del main. La función main también tiene un prototipo, y por lo tanto también puede recibir datos.

```
int main ( int argc, char *argv[], char *envp[] )
```

El parámetro argc es un entero que contiene el número de argumentos pasados a la función desde la línea de comandos. Es decir, almacena el número de argumentos en la tabla argv. El parámetro argv es una tabla de cadenas de caracteres (se verá más adelante), cada cadena de caracteres contiene un argumento pasado en la linea de comandos, la primera cadena contiene el nombre con el que fue invocado y las siguientes son los demás argumentos. El parámetro

envp es una tabla de cadenas, que pasa información sobre una variable de entorno del sistema operativo.

Paso de argumentos

Cuando hablamos de argumentos, nos referimos a los valores que aparecen en la llamada a la función. En el ejemplo anterior los argumentos fueron “dato1” y “dato2”. Y los parámetros son “a” y “b”, los que reciben los datos. Notemos que el prototipo de la función tiene los nombres de los parámetros distintos a los de la definición, esto no afectará el comportamiento del programa, pero se vería mejor si fueran iguales. Lo que no puede cambiar es el tipo de datos que va a recibir. En el caso del main, el paso de argumentos se realiza desde la línea de comandos, como ejemplo ponemos el caso de MS-DOS con el comando dir. Si invocamos a este comando escribimos dir /w /p; el argumento argc tiene el valor 3, y argv contiene 3 cadenas de caracteres: dir, /w, /p.

Valores de retorno

Una función puede regresar cualquier tipo de valor excepto tablas u otras funciones. Esta limitación podría resolverse utilizando estructuras o punteros, pero se verá más adelante.

Macros

Una macro es una parte del código que puede parecer y actuar como una función, se define después de las librerías mediante un #define. Se denominan instrucciones de preproceso, porque se ejecutan al comienzo de la compilación. Sin embargo, no hay que abusar de ellas, porque podrían entorpecer el código y hacer lento el programa.

```
#include <iostream.h> #define
mayor(a,b) (a>b)? a: b
int main(){
int a,b; cout<<"teclea 2 numeros
distintos"<<endl; cin>>a; cin>>b; cout<<"el
mayor de esos numeros es: "
<<(mayor(a,b))<<endl;
cin.ignore(); cin.get();
return 0;
}
```

Debemos tener cuidado si vamos a usar macros, las macros, al compilar el programa, se sustituyen directamente en donde se invocan, es decir, que en este ejemplo, es como si se introdujera directamente los condicionales dentro del cout. Así, debemos tener cuidado en introducir bloques grandes de código en una macro (algo nada recomendable), sobre todo, cuidado con los puntos y comas. Podemos ver que las macros al igual que las funciones pueden tener parámetros, sin embargo, tienen que escribirse con cuidado. Por ejemplo, al hacer una macro de la siguiente manera:

#define curva(a) 1+2*a+a*a si en nuestro código colocamos una instrucción curva(1+3) se sustituye por 1+2*1+3+1+3*1+3 contrario a lo que esperaríamos si invocáramos curva(4), 13 en lugar de 25. Habría que definir la macro como 1+2*(a)+(a)*(a). El uso de macros hace más difícil la depuración de un programa, porque en caso de haber errores en la macro, el compilador nos avisará que hay errores, pero en la implementación de la macro. Además de que no realizará comprobaciones de tipo de datos ni conversiones.

En el lenguaje C, se acostumbra usar macros para definir constantes (porque a diferencia de C++, ahí no existe el tipo const.

Por ejemplo:

#define PI 3.141592 cambiará todas las apariciones de la palabra "PI" por el valor 3.141592.

Recursividad

La recursividad se presenta cuando una función se invoca a si misma. Distintamente a las iteraciones (bucles), las funciones recursivas consumen muchos recursos de memoria y tiempo. Una función recursiva se programa simplemente para resolver los casos más sencillos, cuando se llama a una función con un caso más complicado, se divide el problema en dos partes, la parte que se resuelve inmediatamente y la que necesita de más pasos, ésta última se manda de nuevo a la función, que a su ves la divide de nuevo, y así sucesivamente hasta que se llegue al caso base. Cuando se llega al final de la serie de llamadas, va recorriendo el camino de regreso, hasta que por fin, presenta el resultado. Un ejemplo clásico para este problema es calcular el factorial de un número, (n!). Conocemos los casos base, 0!=1, 1!=1. Sabemos que la función factorial puede definirse como n!= n·ðß n-1ðà ! . Entonces, 5! = 5*4! = 5*4*3! = 5*4*3*2! = 5*4*3*2*1! = 120.

Nuestro programa quedaría como sigue:

```
#include <iostream.h>
long factorial(long numero); //prototipo int main(){
long numero; cout<<"número para calcular el
factorial:"<<endl; cin>>numero; cout<<"el factorial
es: "<<factorial(numero)<<endl; cin.ignore();
cin.get();
return 0;
```

```
}
long factorial(long numero){
if(numero<=1) return 1; else return
numero*factorial(numero1);
}
```

Nuestro programa llama a la función factorial con un número inicial, posteriormente, esta función se llama a sí misma cada vez con número más pequeño hasta que llega al caso base (numero<=1).

La recursividad es un tema importante en el mundo de la programación, la utilidad de este tipo de funciones se aprovecha en el diseño de algoritmos de criptografía, de búsqueda, entre otros. La implementación de la recursividad en nuestros programas debe de evaluarse con mucho cuidado, debemos de tratar de evitar que se usen funciones de complejidad exponencial, que se llamen a sí mismas una y otra vez sin que se tenga un control claro.

Capítulo 8 estructuras

En muchos lenguajes, se le permite al programador definir sus propios tipos de datos a partir de los tipos atómicos. En C y C++ pasa lo mismo, y además existen de antemano los tipos union, enumeración y estructura. Estos tipos definidos forman parte del estandar de C, y por consiguientede C++, en éste último existe también el tipo class.

Para los objetivos de este trabajo excluiremos el tipo class (clase), pero a lo largo del capítulo mencionaremos en varias ocasiones este concepto para tener una idea de lo que se trata.

Enumeraciones

En ocasiones habremos de declarar constantes enteras una tras otra con valores consecutivos, por ejemplo: const int a=0,b=1,c=2,d=3; para evitar este tipo de declaraciones una tras otra, existe el tipo enumerativo. La declaración anterior se puede hacer de una mejor forma:

enum {a,b,c,d};

Una enumeración puede ser global o local. Además de que puede tener un nombre que los identifique, podemos indicar desde donde empieza la numeración. enum meses { enero=1, febrero, marzo, abril, mayo, junio, julio, agosto, septiembre, octubre, noviembre, diciembre}; Los valores que se le asignarán empiezan por 1 a enero, 2 a febrero y así sucesivamente. Si se le asignase un 5 a marzo, entonces abril tendría un 6 y así continuaría la numeración.

Después de hacer esta enumeración, se puede hacer una nueva a partir del nombre.

meses calendario;

Un ejemplo sencillo ilustrará nuestras ideas.

```
#include<iostream.h>                enum
colores{rojo,amarillo,verde}; int main(){
int  i=0,contador=0;  //unos  contadores
colores semaforo;
cout<<"este programa mostrara el tipo enumerativo,"
<<" por medio de un semaforo de crucero"
<<"          simulado"<<endl;          while(contador<3){
semaforo=verde; cout<<"semaforo en verde, el peaton
no pasa"<<endl; cout<<"el peaton pide el paso con un
boton"
<<"(presione enter)"<<endl;
cin.get();  semaforo=amarillo;  cout<<"semaforo  en
amarillo,  precaucion"<<endl;  for(i=0;i<500000000;i++);
```

```
semaforo=rojo; cout<<"semaforo en rojo, ALTO, pasa el
peaton"<<endl; for(i=0;i<500000000;i++);
87 contador++;
}
return 0;
}
```

Debemos tomar en cuenta que al declarar un tipo de variable en base a una enumeración, en este caso una variable semaforo de tipo colores, sólo se podrá asignar valores declarados dentro de esta, para el ejemplo anterior sólo puede tomar valores rojo amarillo o verde.

Uniones

El tipo de datos union puede contener datos de tipos y tamaños diferentes, esta variable almacenará cualquier tipo de dato en una única localidad en memoria. Por ejemplo:

```
union edificio{ int
habitaciones; int
despachos;
char empresa[10];
};
```

declara un nuevo tipo de dato llamado “edificio”, para usar una variable de tipo edificio bastará con hacer una declaración como:

edificio casa; la variable casa tendrá el suficiente espacio en memoria para almacenar una cadena (el tipo de dato de mayor longitud dentro de la union). Es responsabilidad del programador extraer adecuadamente el tipo de dato deseado.

La forma en que está estructurada en la memoria nuestra variable es como la que sigue: Byte1 Byte2 Byte3 Byte4 Byte5 Byte6 Byte7 Byte8 Byte9 Byte10 Espacio para las variables “habitaciones” y “despachos” Espacio para la cadena “empresa” Cuando se almacena un entero y posteriormente una cadena, ésta última se escribe sobre el espacio de memoria asignado al entero, y por lo tanto, el contenido se pierde.

Estructuras

Las estructuras son colecciones de elementos, los cuales pueden ser de diferente tipo, y a diferencia de las uniones, en éstas no se comparte la misma área de memoria para los elementos. La forma de declarar una estructura es más o menos como sigue:

```
struct nombre_estructura{ tipo
nombre_elemento1;        tipo
nombre_elemento2;
};
```

Es entonces cuando podemos declarar una variable del tipo de la estructura, funcionando el nombre de la estructura como un nuevo tipo de datos, de esta manera, la nueva variable tendrá cada uno de los elementos declarados en ella. nombre_estructura nueva_variable;

Para acceder a cada uno de sus elementos usamos el operador punto (.), seguido del nombre del nombre del elemento, de la misma forma que con las uniones. Una estructura puede ser declarada de forma global o local, pero es más común hacerlo fuera de toda función.

Capítulo 9 entrada y salida por archivos

La entrada y salida por archivos es uno de los temas más importantes para la programación, es necesario para poder programar desde una pequeña agenda

hasta una completa base de datos, pero antes de entrar de lleno a este tema, y aprovechando los conocimientos adquiridos hasta el momento, quiero mostrar el uso de la memoria dinámica.

Memoria dinámica

Hasta el momento no contamos con una forma de “administrar” la memoria utilizada en nuestros programas, cuando declaramos una variable se asigna memoria para almacenar datos dentro de ella, y ésta no se destruye hasta que termina el bloque en el que fue declarada, se crea y se destruye cada que se pasa por ese bloque, quizá puede parecer poco importante, pero cuando se cuentan con grandes cantidades de datos, es necesario tener un mejor control de lo que se pone en memoria. Una de las formas en que podemos asegurarnos que una variable declarada dentro de un bloque no sea borrada al término de éste, es mediante la utilización del calificador static. De ésta manera, la variable perdurará hasta el término de todo el programa.

static tipo_variable mi_variable;

Pero en algunos casos esto nos será insuficiente, vamos a necesitar “destruir” variables antes de terminar el programa, para hacer esto contamos con los operadores new y delete. Se puede utilizar el operador new para crear cualquier tipo de variables, en todos los casos devuelve un puntero a la variable creada. En el momento en que se quiera borrar esta variable deberemos utilizar el operador delete, todas las variables creadas mediante el operador new deben de ser borradas con el otro operador. En el siguiente ejemplo utilizaremos estos dos operadores para mostrar sus ventajas.

```
#include<iostream.h>
int main(){
char *cadena2; int fincontador=10;
cout<<"programa de prueba"<<endl;
for(int i=0;i<fincontador;++i){
cout<<"contador funcionando "<<i<<endl;
}
int respuesta; cout<<"crear
variable?, 1 para si";
cin>>respuesta; char
cadena[3]="hi";
if(respuesta==1){ int
LongCad;
cout<<"variable creada"<<endl;
cout<<"longitud de cadena: ";
cin>>LongCad; cadena2=new
char[LongCad]; cout<<"escribe:
"<<endl;
cin.ignore();
cin.getline(cadena2,LongCad);
cout<<"cadena escrita: "<<cadena2<<endl;
}else{
char cadena[10]="rayos, no";
}
cout<<cadena<<endl;
97
cout<<cadena2<<endl;
delete cadena2;
cin.ignore(); cin.get();
return 0;
```

}

Las partes que nos interesan de este programa son la declaración de la variable tipo apuntador que se encuentra al principio del main, la asignación de espacio dentro del bloque if, y por último la destrucción de nuestra variable lineas antes del término del programa. Si hubiésemos declarado el apuntador dentro del bloque if, cuando se saliera de éste entonces la variable ya no existiría y habría un error en tiempo de compilación. El uso de la memoria dinámica nos será muy útil en muchos casos, en otros quizá prefiramos hacerlo de la manera más común. Estos operadores funcionan tanto para variables simples como para compuestas (definidas por el programador). De nueva cuenta debo mencionar a las clases, porque en éstas se maneja mucho este concepto, y de nuevo invito al lector a que, terminando esta lectura, se adentre más en el mundo de la programación consultando otra bibliografía. Si se anima a trabajar con clases lo felicito, antes quisiera darle una idea manejando estructuras con funciones. El siguiente programa es una modificación del último que vimos en el capítulo anterior, la diferencia está en el manejo de los operadores descritos hace unos párrafos, y por ende en el manejo de apuntadores a estructuras.

```
#include<iostream.h>
#include<iomanip.h> const int NumMaterias=11; struct alumno {
int boleta, semestre; int calificaciones[NumMaterias]; void constancia(alumno * este){
```

98

```
cout<<"constancia impresa"<<endl;
cout<<"alumno con boleta: "<<este> boleta;
cout<<setw(50)<<"del semestre numero: "
<<este> semestre<<endl; cout<<"\nCALIFICACIONES
DEL SEMESTRE: "<<endl; for(int
i=0;i<NumMaterias;++i){ cout<<"asignatura "<<i+1<<" : "
<<este> calificaciones[i]<<endl;
}
}
void inicia(alumno * este){ for(int
i=0;i<NumMaterias;++i) este>
calificaciones[i]=0; este>
boleta=0; este> semestre=0;
} };
int main(){ alumno *alan =new
alumno; alan>
inicia(alan); cout<<"alan
creado"<<endl; alan>
boleta=2005630170; alan>
semestre=5; alan>
constancia(alan);
delete alan; alumno
*alfonse=new alumno; alfonse>
inicia(alfonse);
alfonse>
constancia(alfonse);
delete alfonse; alan>
constancia(alan);
cin.get();
return 0;
```

}

Como en el ejemplo anterior, declaramos una variable de tipo apuntador (apuntador a alumno) y luego le asignamos espacio en memoria. Cuando se invoca a la función miembro “inicia”, se envía como argumento el nombre del apuntador, y como este contiene una dirección de memoria, pues entonces la función que lo recibe esta preparada para recibirlo y manejarlo adecuadamente con el operador flecha.

Note que después de la destrucción de la variable “alan” se crea otra de nombre “alfonse” que también se destruye, y posteriormente se invoca de nuevo a la función “constancia” de “alan”, dependiendo del compilador utilizado puede haber distintos resultados, pero es lógico que no se obtendrá lo que deseamos porque esa variable ya no existe.

ARCHIVOS

Para empezar con el manejo de archivos es necesario recordar el concepto de flujo, el cual se define como un dispositivo que consume o produce información. En nuestros programas hechos hasta el momento hemos utilizado los flujos stándar cin, cout y cerr, el resto de los flujos que se deseen deberán ser creados por el programador. Todos los flujos se comportan de forma análoga, independientemente del dispositivo que se trate.

Para poder usar un flujo estándar basta con incluir la biblioteca iostream.h como lo hemos hecho hasta ahora. Cuando decidimos utilizar la función cin.get() no sabíamos exactamente el porque de la utilización del punto(.), ahora que hemos visto un poco el concepto de estructura podemos decir que se trata de la invocación a una función miembro del flujo cin. Sí, el flujo es un objeto, viéndolo desde la perspectiva de las estructuras y no precisamente de las clases como debería de ser, se trata de un tipo de dato que contiene variables y funciones que pueden ser invocadas. A lo largo de este capítulo hablaremos de los flujos como estructuras, y no como clases para tener una mejor idea de lo que se tratan.

Las estructuras (en realidad clases) para la entrada y salida de datos tienen un orden jerárquico, en ellas existe la herencia que básicamente consiste en que una de orden inferior obtiene las mismas variables y funciones que la de orden mayor, además de que puede agregar más. Para poder trabajar los ficheros como flujos es necesario incluir la librería fstream.h, y según la utilización que queramos dar a este fichero (lectura o escritura) deberemos declarar el tipo de flujo.

Para crear un archivo de salida declaramos una variable de tipo ofstream, el cual ya está declarado dentro de nuestra librería. Es mejor ver un ejemplo de base. #include<fstream.h> int main(){ ofstream archivo; archivo.open("miarchivo.txt"); archivo<<"hola desde este archivo\n"; archivo<<"ya he escrito algo\n"; archivo.close();

}

Aquí declaramos a “archivo” como variable tipo ofstream, y posteriormente utilizamos su función miembro open para asociarla a un archivo, se pude asociar directamente en la declaración de la siguiente manera:

101 ios

istrstream

istream

istream_withasign ifstream

ostream

ofstream ofstream_withasign

ostrstream iostream fstream strstream

stdiostream ofstream

archivo(“miarchivo.txt”);

Tanto la primera como la segunda forma admiten un segundo argumento que especifica el modo de apertura de un archivo. Los modos disponibles se muestran en la siguiente tabla y pueden ser utilizados incluyendo la librería iostream.h.

ios::app Se escribe al final de archivo ios::out
El archivo se abre para escritura
ios::trunc Si el archivo existe se eliminará su contenido ios::in El archivo se abre para lectura, el archivo original no será modificado ios::binary El archivo se abre en modo binario

Tabla 12 Modos de apertura de archivo

Con el archivo creado en el ejemplo anterior utilizaremos el siguiente programa para escribir al final de él.

```
#include<iostream.h> #include<fstream.h>
int main(){
ofstream archivo("miarchivo.txt", ios::app); archivo<<"hola
desde este archivo de nuevo\n"; archivo<<"ya he escrito
algo de nuevo\n";
archivo.close();
}
```

El método para abrir un archivo en modo lectura es muy similar, pero en este caso utilizaremos ifstream. Para tener el control del fichero, aparte de conocer los modos de apertura de un archivo, debemos de conocer el delimitador, así como en las cadenas existe el carácter de fin de cadena('\0'), en los archivos está el fin de archivo (EOF). El siguiente programa lee caracteres de un archivo y los imprime en pantalla hasta llegar al fin de éste.

102

```
#include<iostream.h> #include<fstream.h>
int main(){ char
caracter;
ifstream archivo("miarchivo.txt", ios::in);
while(!archivo.eof()){ archivo.get(caracter);
```

```
cout<<caracter;
}
archivo.close();
}
```

El programa abre el archivo en modo de lectura, inmediatamente el indicador de posición se coloca en el primer caracter del flujo (el archivo), la función eof() devuelve verdadero en caso de que en la posición en la que está el indicador esté el fin de archivo, nosotros hacemos una comprobación de ésto, y mientras (bucle while) no se llegue al final de archivo se leerá un caracter de éste flujo, al hacer esto el indicador de posición avanzará, posteriormente se imprime el caracter leído en pantalla y continua el ciclo. ¿Que pasaría si el archivo no existiese?, el programa entraría en un ciclo infinito, por eso debemos de asegurarnos de que el flujo se ha creado bien.

```
#include<iostream.h> #include<fstream.h>
int main(){ char caracter; ifstream
archivo("miarchivo2.txt", ios::in);
if(archivo){ while(!archivo.eof()){
archivo.get(caracter);
cout<<caracter;
}
archivo.close();
}else{
cerr<<"el archivo no existe"<<endl;; return
1;
103
}
return 0;
}
```

Para el manejo de caracteres desde un archivo podemos utilizar las funciones miembro get, getline, read, write, ignore, gcount con las que ya tenemos un poco de experiencia. Además de éstas, existen otras funciones que nos serán muy útiles para que no sea tan secuencial la forma en la que leemos o escribimos el archivo. tellg() Devuelve la posición de lectura seekg() Se coloca dentro del flujo en la posición pasada como argumento para poder leer. tellp() Devuelve la posición de escritura seekp() Se coloca dentro del flujo en la posición pasada como argumento para poder escribir.

Ya sabrá de la importancia de los archivos, pero quizá piensa en cosas demasiado grandes, pensar en una base de datos aveces puede causar que nuestro interés se deteriore con el avance del proyecto, esto claro, si no se tiene claramente definido el objetivo de ésta. Para practicar con el manejo de archivos piense en algo pequeño, algo que le llame la atención, vuelva a recordar que con cosas pequeñas se construyen cosas más grandes. Hasta aquí el tema de manejo de archivos creo que nos ha dado las suficientes herramientas para desarrollar cosas mejores. ¡¡Ohh, la computadora tiene un espíritu dentro!!!, quizá somos muchas las personas que hemos visto a alguien decir eso, en el bachillerato llegué a ver varias personas un poco asustadas por ello. Para practicar un poco más, y utilizar algunas funciones vistas hasta ahora le muestro el siguiente programa.

```
#include<iostream.h>
#include<fstream.h> #include<string.h> int main(){
char
texto_preg[300],texto_resp[300],*compara=NULL;
ifstream archivo("arch.txt", ios::in);
if(!archivo){
cerr<<"error al abrir el archivo";
return 1;
}
```

```
cout<<"* hola"<<endl; cout<<"> ";
cin.getline(texto_resp,290);
while(texto_resp[0]!='\0'){
archivo.seekg(0);                    do{
archivo.getline(texto_preg,290);
compara=strstr(texto_resp,texto_pre
g);
if(archivo.eof()){ cout<<"* no se contestar,
dimelo por favor"<<endl;
cin.getline(texto_preg,290); archivo.close();
ofstream archivoS("arch.txt",ios::app);
archivoS<<texto_resp<<endl;
archivoS<<texto_preg<<endl;
archivoS.close();
archivo.open("arch.txt", ios::in);
}
}while(compara==NULL);
cout<<"> ";
archivo.getline(texto_preg,290);
cout<<"*          "<<texto_preg<<endl;
cout<<"> ";
cin.getline(texto_resp,290);
}
archivo.close();
return 0;
}
```

Se trata de un simple programa de conversación, donde el elemento principal es el archivo “arch.txt”, el cual contiene una conversación en él:

105 hola como
estas? bien
que haces?
nada pues ponte a hacer algo, a que te dedicas?
estudio y que
estudias?
informatica ha que bueno!!, de
donde eres?
ixtapaluca que interesante, hay algo que quieras preguntarme? como te llamas? acaso eso importa?
como te llamas
pregunta de nuevo

El programa inicia abriendo el archivo en modo lectura, si existe un error el programa termina. Empieza su conversación diciendo “hola”, y espera a que el usuario responda, atrapa su respuesta y la guarda en una cadena, si tecleó algo que no fuera simplemente un enter entonces continua dentro del bucle.

Se posiciona al inicio del archivo, y, dentro de otro bucle, lee la cadena de caracteres del archivo con un máximo de 290 caracteres y la almacena en “texto_preg”. Utilizando la función strstr() de la biblioteca string.h, compara la cadena introducida por el usuario con la cadena que acaba de leer del archivo, la función strstr() devuelve un apuntador a la primera localización de la cadena “texto_preg” en “texto_resp”, si no se encuentra devuelve NULL.

Si se ha llegado al final del archivo, es decir, no se encontró que contestar, pide a la persona que le diga que debe de contestar, entonces cierra el archivo y lo vuelve a abrir pero esta vez en modo de escritura para añadir la pregunta del usuario con su respectiva respuesta, luego cierra el archivo de salida y lo abre en modo de lectura de nuevo. Estos pasos de buscar y, si no encuentra entonces preguntar,

continuarán mientras no se encuentren coincidencias entre lo escrito y lo que está dentro del archivo. Después, continuando con el bucle de mayor nivel, vuelve a leer del archivo una cadena de caracteres, y empieza la conversación de nuevo. Para terminar el programa simplemente el usuario debe teclear enter sin ningún otro caracter. Como ya se habrá dado cuenta, la parte principal de este programa es el archivo, en él está la base de la conversación, y si se quiere una plática más amena pues hay que modificar el archivo y aumentar su repertorio de respuestas. Note que al final del archivo hay una linea en blanco para el correcto funcionamiento del programa.

A día de hoy ¿Qué tan importante es C++?

A día de hoy C++ es uno de los mejores lenguajes en el uso eficiente de los recursos y lo mejor de todo es que esta cualidad no se ve mermada si la complejidad del proyecto es bastante alta, así que para escribir software complejo con un buen rendimiento lo mejor es C++ tanto así que proyectos como los siguientes están construidos en C++:

AREAS	APLICATIVOS
Ofimática	Microsoft Office, OpenOffice
Bases de datos	Oracle, MySQL, Micorosft SQL Server
Navegadores de Internet	Internet Explorer, Mozilla Firefox,Safari, Google Chrome, Opera
Otros	Google, PayPal, Amadeus, Amazon, Facebook o DropBox.

Tabla 1 Áreas aplicables para el lenguaje C++

3.2.2 Historia de Microsoft Excel

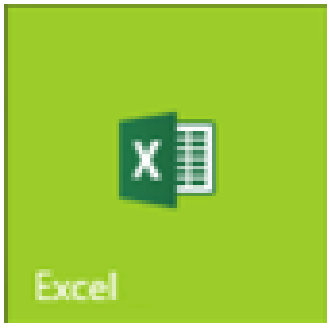


Imagen 3. Microsoft Excel
Logotipo actual

Microsoft Excel, es una aplicación para manejar hojas de cálculo. Este programa fue y sigue siendo desarrollado y distribuido por Microsoft, y es utilizado normalmente en tareas financieras y contables.

Microsoft comercializó originalmente un programa de hoja de cálculo llamado Multiplan en 1982, que se convirtió muy popular en los sistemas CP/M , pero en los sistemas MS-DOS perdió popularidad a Lotus 1-2-3. Microsoft publicó la primera versión de Excel para Mac en 1985, y la primera versión de Windows (numeradas 2-05 en línea con el Mac y con un paquete de tiempo de ejecución de entorno de Windows) en noviembre de 1987.

Lotus fue lenta al llevar 1-2-3 para Windows y en 1988 Excel había comenzado a vender 1-2-3 y esto ayudó a Microsoft a alcanzar la posición de los principales desarrolladores de software para PC. Este logro, destronando al rey del mundo del software, solidificó a Microsoft como un competidor válido y mostró su futuro de desarrollo del software GUI. Microsoft empujó su ventaja con nuevas versiones regulares, cada dos años. La versión actual para la plataforma Windows es Excel 12, también denominada Microsoft Office Excel 2007. La versión actual para Mac OS X es la plataforma Microsoft Excel 2008.

A principios de 1993, Excel se convirtió en el objetivo de una demanda por otra empresa que ya tenía a la venta de un paquete de software llamado "Excel" en el sector financiero. Como resultado de la controversia Microsoft estaba obligada a hacer referencia al programa como "Microsoft Excel" en todos sus comunicados de prensa oficiales y documentos jurídicos. Sin embargo, con el tiempo esta práctica ha sido ignorada, y Microsoft aclaró definitivamente la cuestión cuando se adquirió

la marca del otro programa. Microsoft también alentó el uso de las letras XL como abreviatura para el programa, mientras que éste ya no es común, el icono del programa en Windows todavía consiste en una estilizada combinación de las dos letras, y la extensión de archivo por defecto del formato Excel es .xls.

Excel ofrece muchas interfaces de usuario ajustadas a las más nuevas hojas de cálculo electrónico, sin embargo, la esencia sigue siendo el mismo que en la hoja de cálculo original, VisiCalc: el programa muestra las celdas organizadas en filas y columnas, y cada celda contiene datos o una fórmula, con relativas o absolutas referencias a otras celdas.

Excel fue la primera hoja de cálculo que permite al usuario definir la apariencia de las hojas de cálculo (las fuentes, atributos de carácter y apariencia de las celdas). También introdujo recomputación inteligente de celdas, donde celdas dependientes de otra celda que ha sido modificada, se actualizan al instante (programas de hoja de cálculo anterior recalculaban la totalidad de los datos todo el tiempo o esperaban para un comando específico del usuario). Excel tiene una amplia capacidad gráfica, y permite a los usuarios realizar la combinación de correspondencia.

Cuando Microsoft primeramente empaquetó Microsoft Word y Microsoft PowerPoint en Microsoft Office en 1993, rediseño las GUIs de las aplicaciones para la coherencia con Excel, el asesino de aplicación en el PC en el momento.

Desde 1993, Excel ha incluido Visual Basic para Aplicaciones (VBA), un lenguaje de programación basado en Visual Basic, que añade la capacidad para automatizar tareas en Excel y para proporcionar las funciones definidas por el usuario (UDF) para su uso en las hojas de trabajo. VBA es una poderosa anexión a la aplicación que, en versiones posteriores, incluye un completo entorno de desarrollo integrado (IDE). La grabación de macros puede producir código VBA para repetir las acciones del usuario, lo que permite la automatización de simples tareas. VBA permite la creación de formularios y controles en la hoja de trabajo para comunicarse con el

usuario. Admite el uso del lenguaje (pero no la creación) de las DLL de ActiveX (COM); versiones posteriores añadieron soporte para los módulos de clase permitiendo el uso de técnicas de programación básicas orientadas a objetos.

La funcionalidad de la automatización proporcionada por VBA causo a Excel convertirse en un objetivo para virus en macros. Este fue un grave problema en el mundo corporativo hasta que los productos antivirus comenzaron a detectar estos virus. Microsoft tomó medidas tardíamente para prevenir el uso indebido mediante la adición de la capacidad para deshabilitar las macros completamente, para permitir las macros al abrir un libro o para confiar en todas las macros firmadas con un certificado de confianza.

Las versiones desde 5.0 a 9.0 de Excel contienen varios huevos de pascua, pero desde la versión 10 Microsoft ha adoptado medidas para eliminar este tipo de características indocumentadas de sus productos.

Versiones

Versiones para Windows	
Año	**Nombre de la versión**
1987	Excel 2.0 para Windows
1990	Excel 3.0
1992	Excel 4.0
1993	Excel 5.0 (Office 4.2 y 4.3, también una versión de 32-bit para Windows NT sólo en PowerPC, DEC Alpha, y MIPS)
1995	Excel para Windows 95 (version 7.0) - incluido en Office 95
1997	Excel 97 - incluido en Office 97 (x86 y también una versión DEC Alpha version).
1999	Excel 2000 (version 9.0) incluido en Office 2000
2001	Excel 2002 (version 10) incluido en Office XP

2003	Excel 2003 (version 11) incluido en Office 2003
2007	Excel 2007 (version 12) incluido en Office 2007
Nota: No existe una versión 1.0 para Windows, ya que la versión de Windows fue introducida simultáneamente en la versión para Mac, que fue la 2.0.	
Nota 2: No existe una versión de Excel 6.0, ya que la versión para Windows 95 fue lanzada con Word 7. Todos los productos de Office 95 & Office 4.X tienen capacidades de OLE - moviendo datos entre distintos programas - y Excel 7 debe mostrar que es contemporáneo con Word 7.	

Tabla 2. Versiones de Microsoft Excel para Windows

Versiones para MAC	
Año	**Nombre de la versión**
1985	Excel 1.0
1988	Excel 1.5
1989	Excel 2.2
1990	Excel 3.0
1992	Excel 4.0
1993	Excel 5.0 (Office 4.X -- versión para Motorola 68000 y la primera para PowerPC).
1998	Excel 8.0 (Office '98)
2000	Excel 9.0 (Office 2001)
2001	Excel 10.0 (Office v. X)
2004	Excel 11.0 (parte de Office 2004 para Mac)
2008	Excel 12.0 (parte de Office 2008 para Mac).

Tabla 3. Versiones de Microsoft Excel para Mac

Versiones para OS/2	
Año	**Nombre de la versión**

1989	Excel 2.2
1990	Excel 2.3
1991	Excel 3.0
1992	EXCEL 6.0

Tabla 4. Versiones de Microsoft Excel para OS/2

s el punto más crítico en la construcción del proyecto de investigación, ya que es aquí donde se encuentra el fundamento científico del estudio de investigación, consiste en realizar una revisión de la literatura sobre el tema, es decir, buscar las fuentes documentales que permitan detectar, extraer y recopilar la información de interés para la solución del problema.

Este marco deberá tener por lo menos 5 fuentes bibliográficas (libros, revistas de divulgación científica, artículos arbitrados, tesis, entre otros). Las cuales deberán ser citados utilizando la norma APA.

3.3 Marco Legal (Si aplica)

Véase Anexo 1

CAPÍTULO 4
DESARROLLO DEL PROYECTO DE ESTADÍA

4.1 Recopilación y organización de la información

Con base en las observaciones realizadas durante los primeros días de la estadía, así como también en la encuesta realizada se observó que uno de los problemas principales además de la falta de herramientas y de equipos, era el tiempo perdido al hacer el registro de los horarios de las estradas y de las salidas de los trabajadores.

Llegando a esta conclusión se decidió elaborar una propuesta de un verificador electrónico del horario de la entrada y de la salida del personal como principal objetivo y como segundo, el apoyar en las actividades de mantenimiento en el área de alumbrado público.

4.2 Análisis de la información

La encuesta antes mencionada (véase el Anexo 5) se aplicó a 20 personas de las diferentes áreas que integran el municipio de Ixhuatlancillo Ver., de las cuales se pueden mencionar

-Personal Directivo

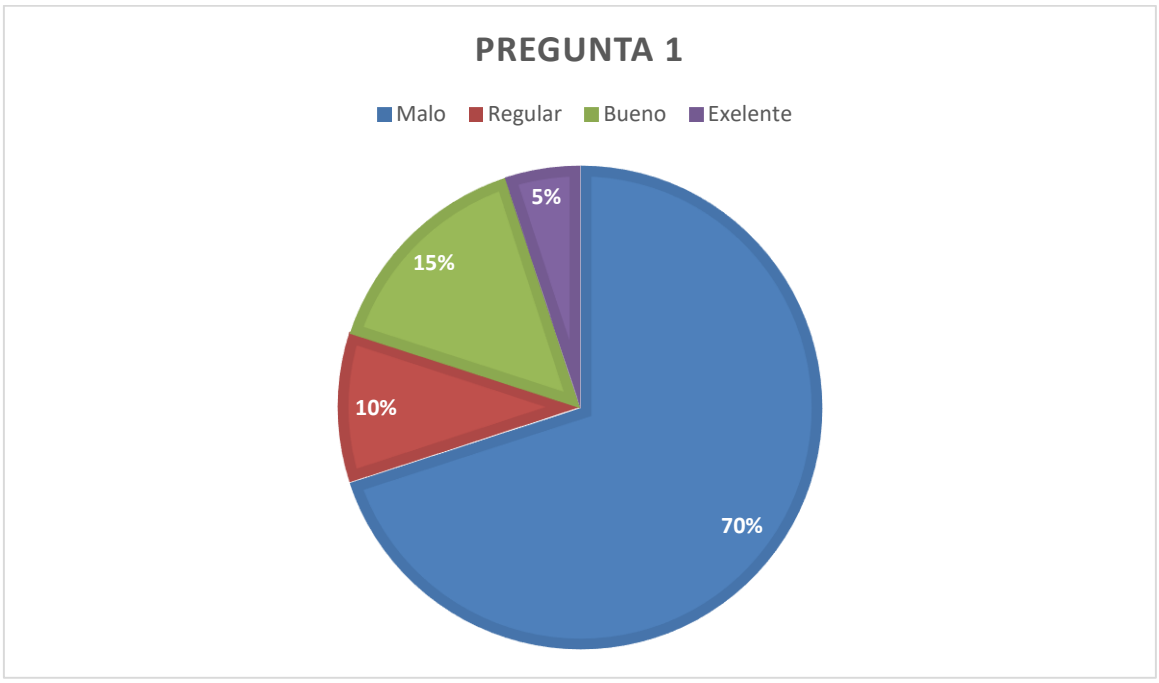
-Personal Administrativo

-Personal Auxiliar

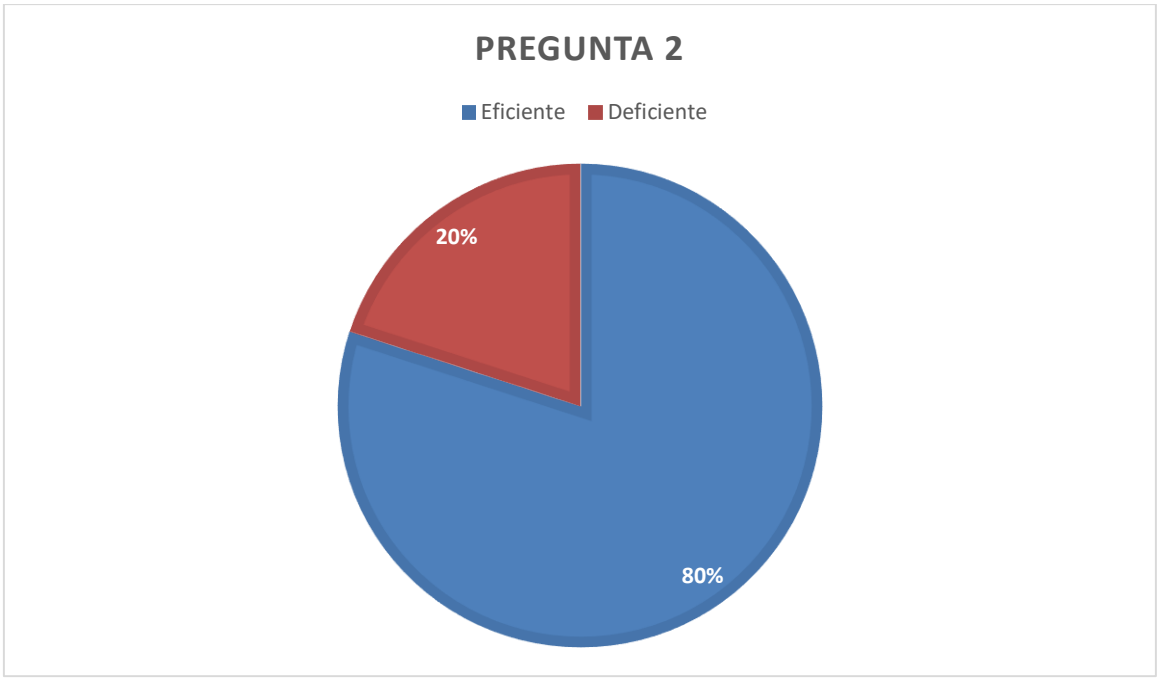
-Personal que labora en campo.

Y nos mostró los siguientes resultados:

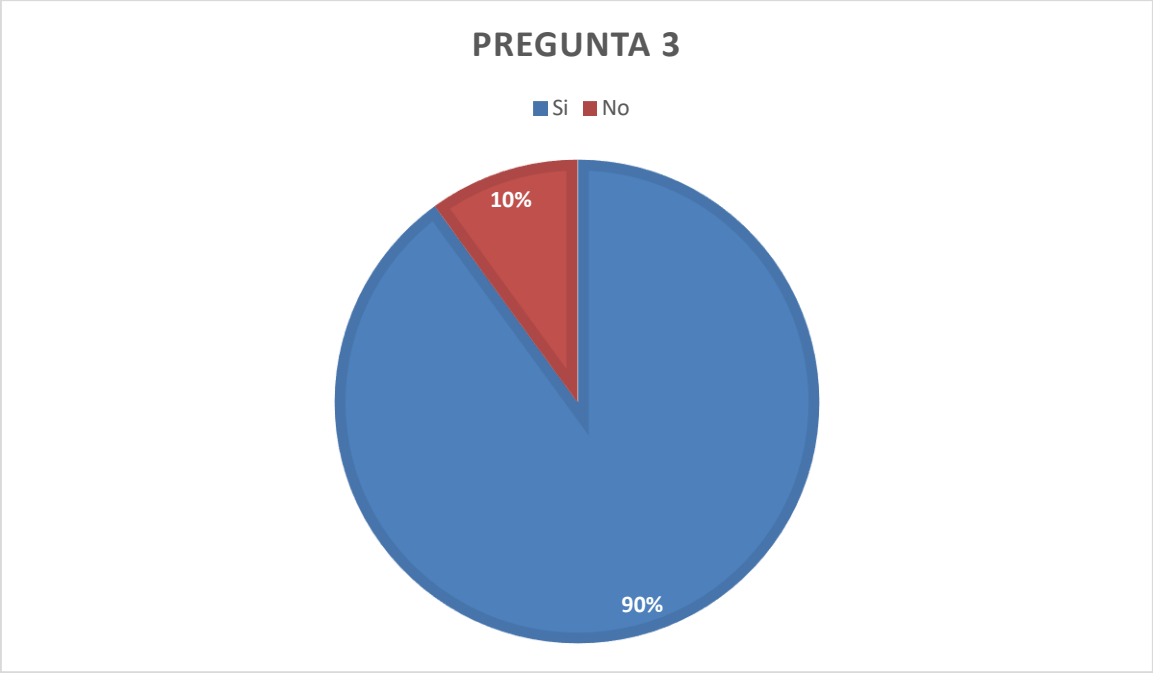
De la primera pregunta: ¿Cómo considera que es el método mediante el cual realiza su registro de presencia?



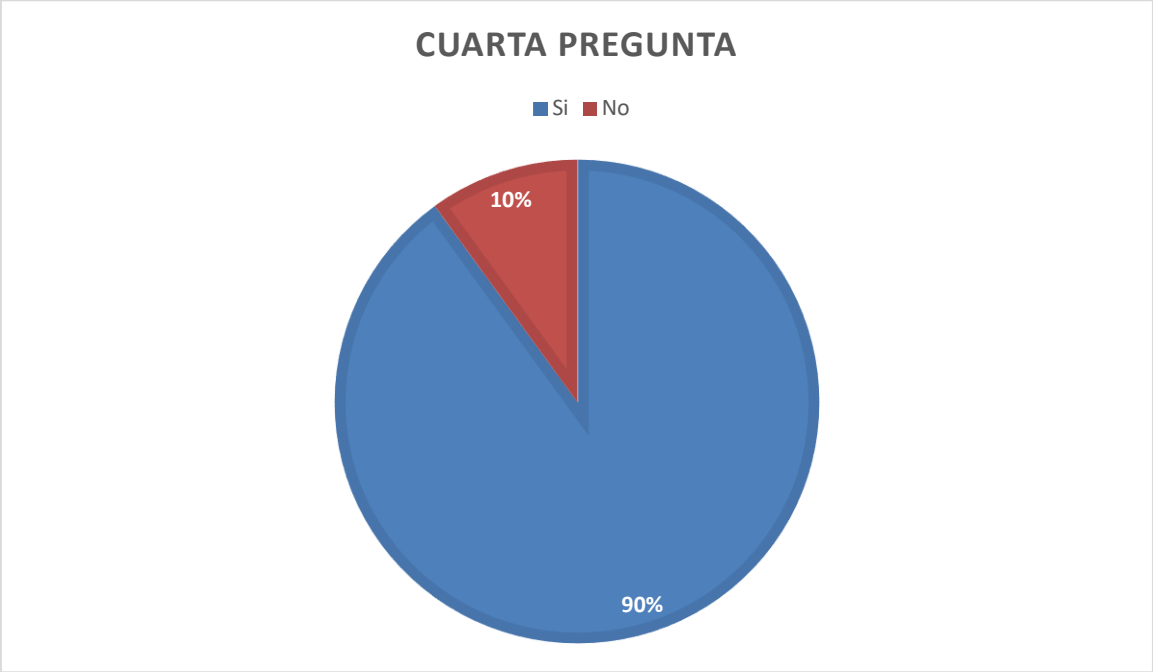
De la segunda pregunta: Considera usted que es un método



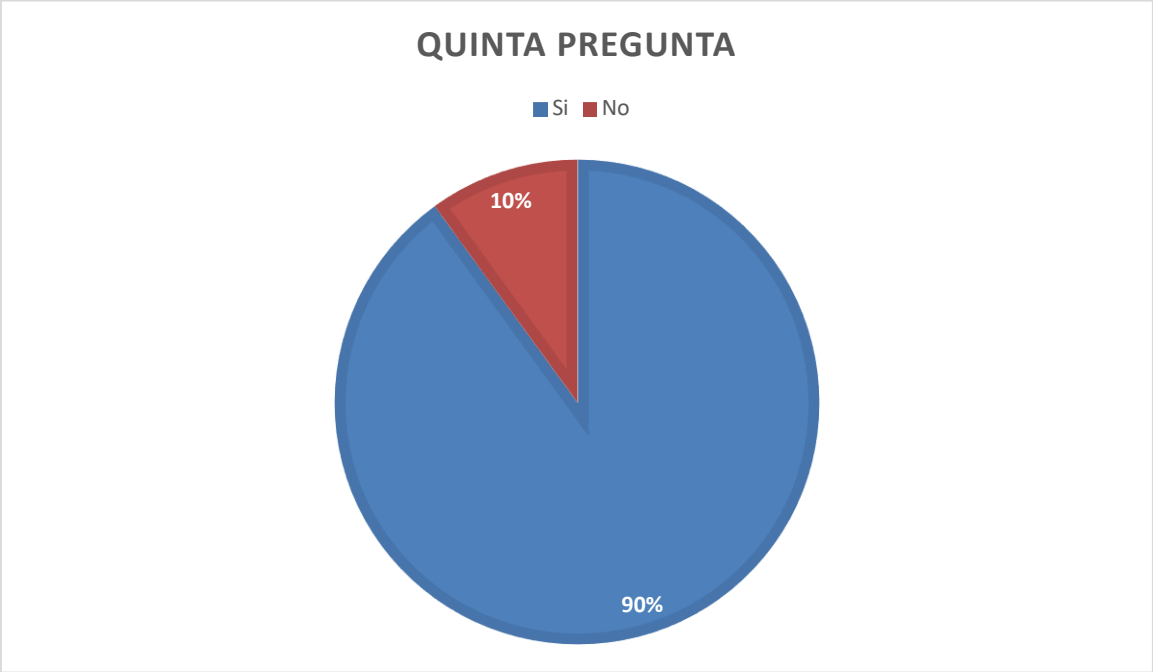
De la tercera pregunta: ¿Cree usted que se pierde tiempo al hacer su registro de presencia?



De la cuarte pregunta: El tiempo que se pierde ¿cree usted que pueda emplearse para realizar actividades de igual o mayor prioridad?



De la última pregunta: ¿Estaría usted a favor emplear un nuevo método digitalizado que sea más eficiente que el método actual y pierda menos tiempo?



Otro tipo de prueba que se realizó fue medir el tiempo que se utilizaba al realizar el registro de entrada de cada trabajador y fue posible observar que se pierden aproximadamente de 40 a 50 minutos que pueden utilizarse para realizar otras actividades de mayor prioridad.

4.3 Propuesta de solución

De acuerdo a los datos obtenidos se llegó a la conclusión que de acuerdo con todo el trabajo que se tenía, ya que recién había comenzado la administración, el tiempo debía de ocuparse sin desperdiciar un solo minuto, por lo que la implementación de un método más eficiente para registrar la entrada y la salida de los trabajadores era prioritario.

Entonces, para dar solución a esta problemática, se sugiere una propuesta que consiste en la creación de un sistema que cumpla la necesidad de un mejor control de los horarios de las entradas y de las salidas del personal y que éste a su vez, emplee herramientas tecnológicas actuales teniendo dos opciones

aplicables a estas necesidades.

4.4 Desarrollo del proyecto

Semana 1

Búsqueda de problemáticas.

La primera semana tuvo como objetivo el buscar en conjunto con el

Semana 2

Búsqueda de problemáticas y apoyo en actividades de mantenimiento

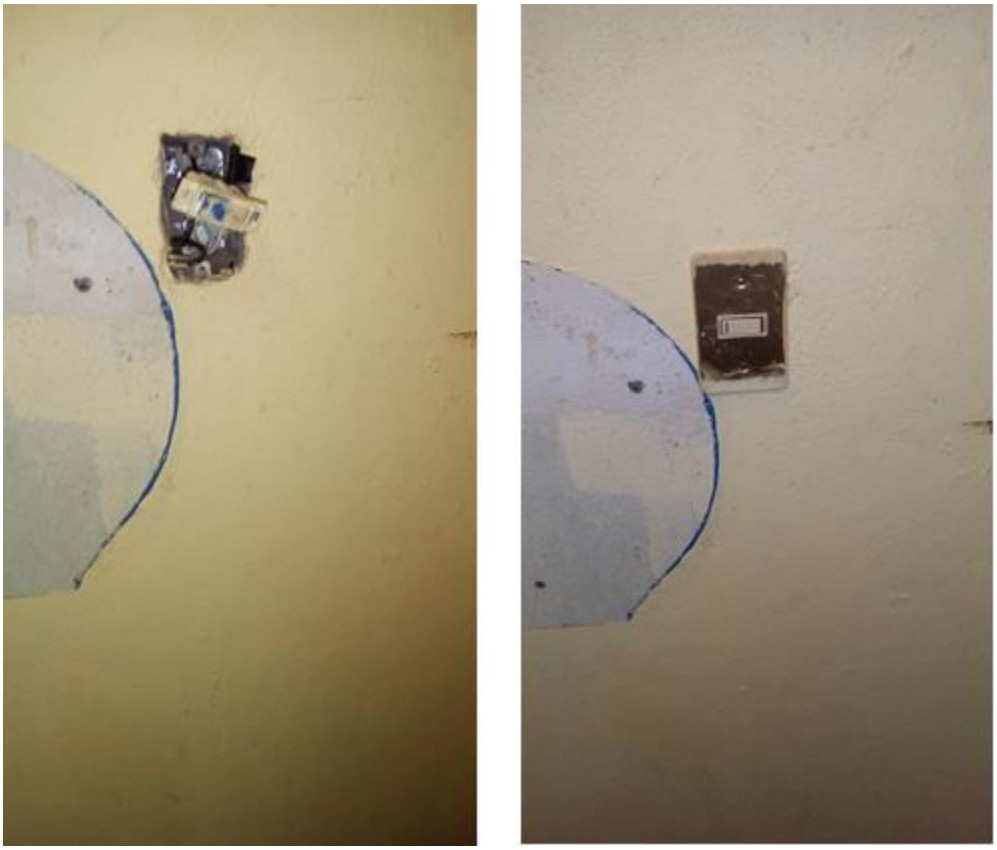


Imagen 4. Apagador antes y después de aplicarle mantenimiento.

Semana 3

Redacción de objetivo introducción y objetivos específicos, apoyo en actividades de mantenimiento

Imagen 5. Ensamble de ménsulas a receptáculos para luminarias

Semana 4

Redacción de metas, alcances, limitaciones, apoyo en actividades de mantenimiento

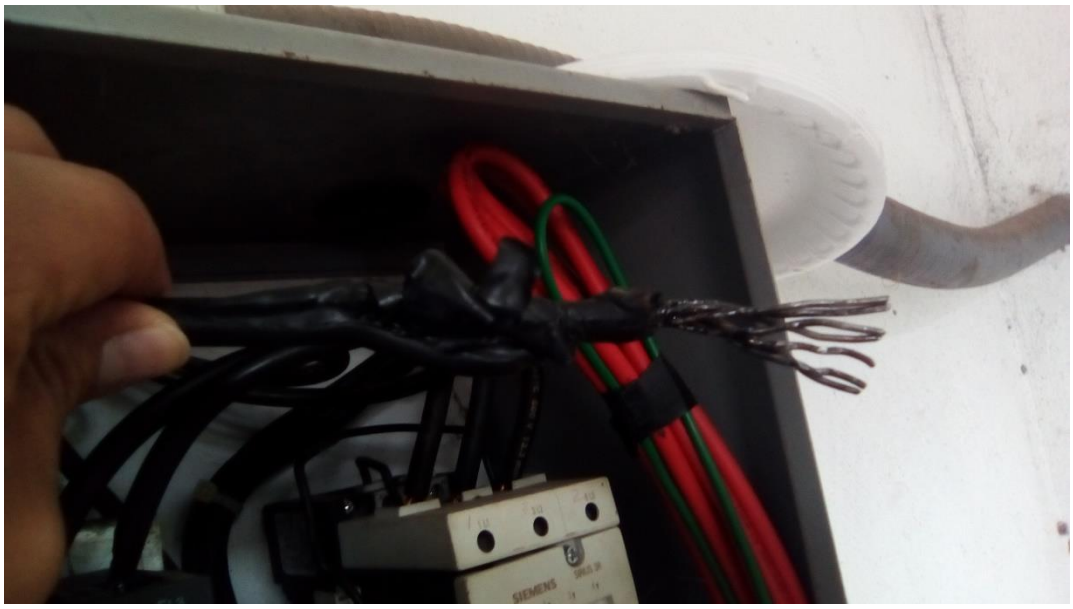


Imagen 6. Cambio de conductores del arrancador de la bomba ubicada en la U.H. Olivos

Semana 5

Investigaciones de marco teórico, instalación eléctrica de la feria municipal

Imagen 7. Instalación de luminarias para la feria municipal

Semana 6

Investigaciones de marco teórico, instalación de la feria municipal

Semana 7

Investigación de costos de la propuesta, desinstalación eléctrica de la feria municipal

Semana 8

Cotización de costos de proyecto. Comienzo de la creación del código

Semana 9

Desarrollo del código de la propuesta, apoyo en actividades de mantenimiento al alumbrado público

Semana 10

Realización de pruebas del código. Apoyo en actividades de mantenimiento

Semana 11

Apoyo en actividades de mantenimiento

Semana 12

Presentación de la propuesta

Semana 13

Apoyo en actividades de mantenimiento al alumbrado público

Semana 14

Pruebas a los dos programas ejecutables

Semana 15

Presentación de la propuesta

CAPÍTULO 5
RESULTADOS

5.1 Resultados

Al aplicar este sistema descrito en la propuesta es posible que del tiempo que se desperdicia checando la estrada del personal se recupere aproximadamente un 80% (aproximadamente 30 minutos); mismo que puede emplearse realizando otras actividades de mayor importancia, sobretodo para los trabajadores que salen a campo diariamente.

CONCLUSIONES

A lo largo de las 15 semanas de estadía en el H. Ayuntamiento Constitucional de Ixhuatlancillo Ver., se logró el objetivo planeado, así como el presenciar y participar en los diversos procedimientos de mantenimiento y la instalación de alumbrado público y sus actividades derivadas.

De acuerdo a lo programado en el cronograma de actividades de la estadía, así como de las actividades asignadas por el asesor industrial dentro de la empresa, se logró realizar los dos códigos planteados en el objetivo de la estadia, asi como también el presenciar y participar los procedimientos de mantenimiento al alumbrado público.

ANEXOS

REFERENCIAS

- **Deitel, Harvey M.; Deitel, Paul J., C++, cómo programar Edit. Pearson Educación. 4ª ed., 2003.**

- **Schildt, Herbert. C, Manual de referencia Edit. Osborne McGraw-Hill. 4ª ed., 2000.**

- **Sierra Urrecho, Alejandro; Alfonseca Moreno, Manuel Programación en C/C++. Edit. Anaya Multimedia. Madrid, España, 1999.**

- **Stevens, Al; Walnum, Clayton.Programación con C++. Edit. Anaya Multimedia. Madrid, España, 2000.**

- **Enciclopedia de las delegaciones y municipios, Estado de Veracruz. Octubre 06 de 1999.** http://siglo.inafed.gob.mx/enciclopedia/EMM30veracruz/municipios/30081a.html

- **Cindy Baine, La historia de Microsoft Excel, Septiembre 05 de 2009.** http://intecforum.forosactivos.net/t93-la-historia-de-microsoft-excel

- **Brian Hudson , La historia de Microsoft Excel,** https://techlandia.com/historia-microsoft-excel-sobre_324527/

- **Luis Zúñiga, La historia del lenguaje C++** https://www.proyectobyte.com/programacion/c/la-historia-del-lenguaje-c-plus-plus