Las plantillas nos permiten especificar, con un solo segmento de código, un rango completo de funciones relacionadas (sobrecargadas), llamadas funciones de plantilla, o un rango completo de clases relacionadas, llamadas clases de plantilla.

Podríamos escribir una sola plantilla de función para una función de ordenamiento de arreglos y luego hacer que C++ generara automáticamente funciones de plantilla separadas que ordenaran un arreglo de int, un arreglo de float, un arreglo de string, etc.

Podríamos escribir una sola plantilla de clase de pila y luego hacer que C++ generara automáticamente clases de plantillas separadas, tales como una clase de pila de int, una clase de pila de fl
oat
, una clase de pila de
string
, etc.

Hay que observar la diferencia entre las plantillas de función y las funciones de plantilla: las plantillas de función y las plantillas de clase son como plantillas con las cuales trazamos formas, y las funciones de plantilla y las clases de plantilla son como los trazos separados, que tienen la misma forma pero pueden trazarse en colores diferentes, por ejemplo.

Plantillas de función

Las funciones sobrecargadas se utilizan normalmente para realizar operaciones similares sobre diferentes tipos de datos. Si las operaciones son idénticas para cada tipo, esto puede realizarse en forma más compacta y conveniente mediante el uso de plantillas de función. Basándose en los tipos de argumento que se proporcionan en las llamadas a esa función, el compilador genera automáticamente funciones de código objeto separadas para manejar adecuadamente cada tipo de llamada. En C esta tarea se puede realizar mediante macros creadas con la directiva de preprocesador #define.

Sin embargo las macros presentan la posibilidad de serios efectos secundarios y no permiten que el compilador realice revisión de tipo. Las plantillas de función proporcionan una solución compacta similar a la de las macros, pero permiten una revisión de tipo completa. Todas las definiciones de plantillas de función comienzan con la palabra clave témplate, seguida de una lista de parámetros formales para dicha plantilla encerrados entre paréntesis angulares (), cada parámetro formal que representa un tipo debe estar precedido por la palabra clave
class
, como en:

```
template <class T >
template <class Element Type >
template <class BorderType , class FillType
```

Los parámetros formales de una definición de plantilla se utilizan (como sucedería con los argumentos de tipos integrados o de tipos definidos por el usuario) para especificar los tipos de argumentos de la función, para especificar el tipo de devolución de la función y para declarar variables en el interior de la función.

Para definir una plantillas de clase, se puede usar el siguiente formato:

```
template <class T>
class Nombre { . . . uso del parámetro T en funcion }
```

Primer programa

El siguiente programa hace uso de plantillas para determinar el mínimo y máximo valor de un arreglo de elementos dado.
La primera función recibe tiene como parámetros un puntero al tipo de elemento dado y el número de elementos y retorna el menor de los elementos que se encuentran en el arreglo.
La segunda función recibe los mismos parámetros y retorna el mayor de los elementos presentes en el arreglo.
Finalmente en la función main, se hace una prueba de estas funciones, con arreglos de enteros y flotantes.

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

template <class T>
T minimo (T * Array, int num
{
   T min = Array [ 0 ] ;

   for ( int i = 1 ;
      if( Array[ i ] < min )
         min = Array [ i ];

   return min ;
} ;

template <class T>
T maximo (T * Array, int num
{
   T max = Array [ 0 ];
```

```
   for (          int i          =          1          ;
      if( Array[  i              ] >        max        )
                  max            = Array [  i          ];

   return         max            ;
};

void main         (              void       )
{
                  int ArrayInt   [          3          ] = {
                  float ArrayFloat [        3          ] = {
                  int i          ;

                  cout           <<         "Arreglo de enteros: "  ;
   for (          i              =          0          ;
                  cout           <<         ArrayInt   [
                  cout           <<         endl       <<
                  <<             "Numero maximo: "      <<         maximo

                  cout           <<         "Arreglo de flotantes: "  ;
   for (          i              =          0          ;
                  cout           <<         ArrayFloat [
                  cout           <<         endl       <<
                  <<             "Numero maximo: "      <<         maximo

                  getch          ();
}
```

Segundo programa

En este programa se hace uso de plantillas, para elaborar una función que permita ordenar los elementos de un arreglo.

Esta función recibe tiene como parámetros, un puntero al tipo de elemento dado, y dos enteros que indican los índices del primero y último elemento.

Aquí se hace uso del algoritmo OrdenarShell para llevar a cabo la tarea. En la función principal se prueba esta plantilla con arreglos de enteros y flotantes.

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

template          <class         T          >
void Ordenar      (              T          *          a
```

```
{
                        int i                   ,                   j                   ,
                        T item_a_ordenar        ;

                        k                       =                   1                   ;

  do
  {
                        k                       =                   3                   *
   } while (            k                       <                   fn                  -

  do
  {
                        k                       /=                  3                   ;

     for (              i                       =                   st                  +
     {
                        item_a_ordenar          =                   a                   [
                        j                       =                   i                   ;

        while (         item_a_ordenar          <                   a                   [
        {
                        a                       [                   j                   ] =
                        j                       -=                  k                   ;

          if(           j                       <                   st                  +
            break;
        }

                        a                       [                   j                   ]=
      }
  } while(              k                       >                   1                   );
}

void main               (                       void                )
{
                        int ArrayInt            [                   3                   ] = {
                        float ArrayFloat        [                   3                   ] = {
                        int i                   ;

                        cout                    <<                  "Enteros: "         <<
                        <<                      "Arreglo original: "    ;

   for (                i                       =                   0                   ;
                        cout                    <<                  ArrayInt            [
```

```
                    Ordenar         (               ArrayInt        ,
                    cout            <<              endl            <<

    for (           i               =               0               ;
                    cout            <<              ArrayInt        [


                    cout            <<              endl            <<
                    <<              "Flotantes: "   <<              endl
                    <<              "Arreglo original: "    ;

    for (           i               =               0               ;
                    cout            <<              ArrayFloat      [

                    Ordenar         (               ArrayFloat      ,
                    cout            <<              endl            <<

    for (           i               =               0               ;
                    cout            <<              ArrayFloat      [


                    getch           () ;
}
```

Tercer programa

En este programa se implementa, mediante el uso de plantillas la clase NuevaPila, que consiste en una Pila, en la que se pueden llevar a cabo las operaciones como insertar y eliminar datos de la misma, mostrar en pantalla los datos de la pila.

Esta es una pila estática, con un número predefinido de 10 elementos.
En al función principal, se usa una pila de enteros, flotantes y caracteres para poder llevar a cabo una prueba de la plantilla creada.

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

enum estado_pila    {               OK              ,               LLENA

template            <class          T               >
class               NuevaPila
{
                    int tamanyo     ;
```

```
                    T               *               tabla               ;
                    int cima        ;
                    estado_pila estado  ;

  public:
                    NuevaPila       (               int                 =
        ~           NuevaPila       () {            delete              []
                    void meter      (               T                   );
                    T sacar         ();
                    void visualizar ();
                    int num_elementos ();
                    int leer_tamanyo () { return    tamanyo             ; }
}

template            <class          T               >
NuevaPila           <               T               > ::                NuevaP
{
                    tamanyo         =               tam                 ;
                    tabla           = new           T                   [
                    cima            =               0                   ;
                    estado          =               VACIA               ;
}

template            <class          T               >
void NuevaPila      <               T               > ::                meter
{
  if(               estado          !=              LLENA               )
                    tabla           [               cima                ++]=
   else             cout            <<              "*** Pila llena ***"    ;

   if(              cima            >=              tamanyo             )
                    estado          =               LLENA               ;
   else
                    estado          =               OK                  ;
}

template            <class          T               >
T NuevaPila         <               T               > ::                sacar
{
                    T elemento      =               0                   ;

  if(               estado          !=              VACIA               )
                    elemento        =               tabla               [--
  else              cout            <<              "*** Pila vac¡a ***"    ;

  if(               cima            <=              0                   )
```

```
                    estado          =               VACIA           ;
  else
                    estado          =               OK              ;

  return            elemento        ;
}

template            <class          T               >
void NuevaPila      <               T               > ::            visualiz
{
  for (             int i           =               cima            -
                    cout            <<              tabla           [
}

template            <class          T               >
int NuevaPila       <               T               > ::            num_el
{
  return            cima            ;
}

void main           ()
{
                    cout            <<              "Probando pila de enteros"

                    NuevaPila       <               int             >
                    s1              .               meter           (
                    s1              .               meter           (
                    s1              .               meter           (

                    cout            <<              endl            <<
                    s1              .               visualizar      ();

                    cout            <<              endl            <<
                    <<              s1              .               sacar
                    s1              .               sacar           ();

                    cout            <<              endl            <<
                    NuevaPila       <               float           >
                    s2              .               meter           (
                    s2              .               meter           (
                    s2              .               meter           (

                    cout            <<              endl            <<
                    <<              "Pila: "        ;
                    s2              .               visualizar      ();
                    cout            <<              endl            <<
```

```
                    <<          s2          .           sacar
                    s2          .           sacar       ();

                    cout        <<          endl        <<
                    NuevaPila   <           char        >
                    s3          .           meter       (
                    s3          .           meter       (
                    s3          .           meter       (

                    cout        <<          endl        <<
                    s3          .           visualizar  ();
                    cout        <<          endl        <<
                    <<          s3          .           sacar
                    s3          .           sacar       ();


                    getch       ();
}
```

Cuarto ejemplo

Mediante el uso de plantillas cree una clase Cola, en la que se puedan llevar a cabo operaciones como: encolar, decolar e imprimir los datos miembro. Realice una función controladora para probar el uso de esta clase.

La clase NodoCola, tiene como amiga a la clase Cola.

Se presenta además una función para las opciones del usuario, y que se encarga de realizar las llamadas a las funciones de las clases. Esta

función es llamada desde main.

En la función main se ha usado como ejemplo una cola de enteros, aunque también se pudo haber usado otro tipo de datos como: char, double, y otros.

```
        #include <iostream.h>
#include <conio.h>
#include <assert.h>

template        <class      TIPONODO    >
class           Cola        ;

////////////////////////////////////
// definicion clase NodoCola /////
////////////////////////////////////

template        <class      TIPONODO    >
```

```
class                NodoCola            {
                     TIPONODO dato       ;                  // dato
                     NodoCola            <                  TIPONODO            > *
                     public:
                     NodoCola            (const             TIPONODO            &);
                     TIPONODO getDato    () const;          // devuelve dato del nodo

                     friend              class              Cola                <
};

// constructor
template             <class              TIPONODO           >
NodoCola             <                   TIPONODO           > ::                NodoCo
    :                dato                (                  info                ),

// devuelve una copia del dato que esta en el nodo
template             <class              TIPONODO           >
TIPONODO NodoCola    <                   TIPONODO           > ::                getDato
    return           dato                ;
}

// definicion enumeracion //
enum bool            {                   false              ,                   true

////////////////////////////////////
// definicion clase Cola /////////
////////////////////////////////////

template             <class              TIPONODO           >
class                Cola                {
                     NodoCola            <                  TIPONODO            > *
                     NodoCola            <                  TIPONODO            > *
                     public:
                     Cola                ();                // constructor
                     ~                   Cola               ();                 // destru
                     void encolar        (const             TIPONODO            &);
                     bool decolar        (                  TIPONODO            &);
                     bool estaVacia      () const;          // verifica si la cola esta vacia
                     void imprimir       () const;          // imprime datos de la cola

    // funci¢n de utileria para asignar un nuevo nodo
                     NodoCola            <                  TIPONODO            > *
};

// constructor predeterminado
template             <class              TIPONODO           >
```

```
Cola            <           TIPONODO        > ::            Cola

// destructor
template        <class      TIPONODO        >
Cola            <           TIPONODO        > :: ~          Cola
    if( !       estaVacia   () ) {          // ingresa si la cola no esta vaci
                cout        <<              "Eliminando nodos ..."  <<

                NodoCola    <               TIPONODO        > *

   while (      actual      !=              0               ) {
                temporal    =               actual          ;
                cout        <<              temporal        ->
                actual      =               actual          ->
                delete temporal ;
   }
    }
                cout        <<              "Todos los nodos han sido elim
}

// inserta un nodo
template        <class      TIPONODO        >
void Cola       <           TIPONODO        > ::            encolar
                NodoCola    <               TIPONODO        > *

    if (        estaVacia   () )            // si la cola esta vacia
                primero     =               ultimo          =

    else {      // si la cola no esta vacia
                ultimo      ->              sig             =
                ultimo      =               nuevo           ;
    }
}

// elimina un nodo
template        <class      TIPONODO        >
bool Cola       <           TIPONODO        > ::            decolar
    if(         estaVacia   () )            // la cola esta vacia
                return      false           ;               // elimin

                NodoCola    <               TIPONODO        > *

    if(         primero     ==              ultimo          )
                primero     =               ultimo          =

    else
```

```
                    primero         =               temporal        ->

                    valor           =               temporal        ->
                    delete temporal ;
    return          true            ;               // eliminacion satisfactoria
}

// verifica si la cola esta vacia
template            <class          TIPONODO        >
bool Cola           <               TIPONODO        > ::            estaVac
    if (            primero         ==              0               ) return
    return          false           ;
}

// imprime el contenido de la cola
template            <class          TIPONODO        >
void Cola           <               TIPONODO        > ::            imprimi
    if (            estaVacia       () ) {
                    cout            <<              "La lista esta vacia"   <<
    return;
    }

                    NodoCola        <               TIPONODO        > *

                    cout            <<              "La cola es: "  ;

    while (         actual          !=              0               ) {
                    cout            <<              actual          ->
                    actual          =               actual          ->
    }

                    cout            <<              endl            <<
}

// funcion de utileria: devuelve un apuntador a un nodo recientemente asignado
template            <class          TIPONODO        >
NodoCola            <               TIPONODO        > *             Cola
                    NodoCola        <               TIPONODO        > *
                    assert          (               nuevo           !=
    return          nuevo           ;
}

//////////////////////////////////////////////////////////////////////////

// funcion que prueba una cola
template            <class          T               >
```

```
void probarCola (Cola < T

                 T valor ;
                 int opcion ;

    for (;;){
                 cout << endl <<
                 cout << "Seleccion: " ;
                 cin >> opcion ;

        switch( opcion ){
          case 1 :
                 cout << "Ingrese dato: " ;
                 cin >> valor ;
                 ObjetoCola . encolar (
            break;

          case 2 :
            if( ObjetoCola . decolar (
                 cout << "Dato " <<

            else cout << "No puede eliminar : la cola esta

            break;

          case 3 :
                 ObjetoCola . imprimir ();
            break;
        }
        if( opcion == 4 ) break;
    }
}

// funcion principal
void main (){
                 clrscr ();
                 Cola < int >
                 probarCola ( ColaEnteros );

}
```