

**Ejercicio 1:** Diseña e implementa una pila en la que adicionalmente a las operaciones push y pop, tengas una función min que proporcione el menor elemento en la pila.

Incluimos la biblioteca estándar de entrada y salida, la biblioteca para el tipo de dato bool, y la biblioteca que contiene funciones para la asignación de memoria dinámica.

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
```

Define una estructura llamada Nodo, que contiene un entero (valor) y un puntero a otro nodo (NodoInferior).

```
struct Nodo{
    int valor;
    struct Nodo* NodoInferior;
};
```

Define una estructura “Pila” que contiene un puntero a Nodo llamado cima y un entero cursor. Esta estructura representa una pila.

```
typedef struct{
    struct Nodo* cima;
    int cursor;
}Pila;
```

Declaración de las funciones crearPila(), consultarCima(), apilar(), Desapilar(), vaciarPila(), isEmpty(), MostrarPila() y MenorPila(). Estas funciones se definirán más adelante en el código.

```
Pila* crearPila();
void apilar(Pila *p, struct Nodo dato);
void Desapilar(Pila *p, struct Nodo *datoExtraido);
bool isEmpty(Pila p);
void MostrarPila(Pila *p);
void MenorPila(Pila *p);
void vaciarPila(Pila *p);
```

La función crearPila() crea una instancia de Pila, asigna memoria dinámica para ella, establece cima como NULL y cursor como 0, luego retorna un puntero a la pila creada.

```
Pila* crearPila(){
    Pila* pilaAsignacion=(Pila*) malloc(sizeof(Pila));
    pilaAsignacion->cima=NULL;
    pilaAsignacion->cursor=0;
    return pilaAsignacion;
}
```

La función vaciarPila() recibe un puntero a Pila y desapila todos los elementos de la pila mediante llamadas a la función Desapilar().

```
void vaciarPila(Pila *p){
    while(!isEmpty(*p)){
        struct Nodo datoExtraido;
        Desapilar(p,&datoExtraido);
    }
}
```

La función apilar() recibe un puntero a Pila y un Nodo como parámetros. Crea un nuevo nodo, asigna el valor proporcionado y establece el nodo inferior como la cima actual de la pila. Luego actualiza la cima de la pila con el nuevo nodo y aumenta el contador del cursor.

```
void apilar(Pila *p, struct Nodo dato){
    struct Nodo* nodoUsuario=(struct Nodo*) malloc(sizeof(struct Nodo));
    nodoUsuario->valor=dato.valor;
    nodoUsuario->NodoInferior=p->cima;
    p->cima=nodoUsuario;
    p->cursor++;
}
```

La función Desapilar() recibe un puntero a Pila y un puntero a Nodo como parámetros. Extrae el valor de la cima de la pila, actualiza la cima y el cursor, y libera la memoria del nodo desapilado.

```
void Desapilar(Pila *p, struct Nodo *datoExtraido){
    struct Nodo* aux=p->cima;
    datoExtraido->valor=aux->valor;
    p->cima=aux->NodoInferior;//p->cima=p->cima->NodoInferior
    aux->NodoInferior=NULL;
    p->cursor--;
    free(aux);
}
```

La función isEmpty() recibe una pila como parámetro y verifica si la cima de la pila es NULL. Si es así, la pila está vacía y devuelve true; de lo contrario, devuelve false.

```
bool isEmpty(Pila p){  
    if(p.cima==NULL){  
        return true;  
    } else{  
        return false;  
    }  
}
```

La función MostrarPila() recibe un puntero a Pila y muestra los datos almacenados en la pila iterando a través de los nodos de la pila, mostrando su valor y disminuyendo la cantidad mientras avanza hacia el nodo inferior. Si la pila está vacía, muestra un mensaje indicando que no hay datos.

```
void MostrarPila(Pila* p){  
    printf ("\nDatos en la Pila:\n\n");  
    struct Nodo *aux;  
    aux = p->cima; //Se trabaja con una copia de la Pila "Principal"  
    if (aux == NULL) {  
        printf ("No hay Datos\n");  
    }  
    int cantidad = p->cursor;  
    while (aux) {  
        printf ("%d. %d\n", cantidad, aux->valor);  
        aux = aux->NodoInferior;  
        cantidad--;  
    }  
}
```

La función MenorPila() recibe un puntero a Pila y encuentra el valor menor en la pila. Itera a través de los nodos comenzando desde la cima hasta el último nodo y actualiza el valor de menor si encuentra un valor menor en algún nodo. Luego muestra el valor menor encontrado.

```

void MenorPila(Pila* p){
    struct Nodo *aux;
    aux = p->cima;
    if (aux == NULL) {
        printf ("No hay Datos\n");
    }
    int menor = p->cursor;
    while (aux) {
        if(menor > aux->valor){
            menor=aux->valor;
        }
        aux = aux->NodoInferior;
    }
    printf("El dato menor de la pila es: %d", menor);
}

```

La función main() es la función principal del programa. Aquí se inicia la ejecución del programa, permite al usuario realizar varias operaciones en la pila, como mostrar datos, agregar datos a la pila, remover datos de la pila, mostrar el valor menor de la pila y salir del programa. El programa se ejecuta en un bucle do-while hasta que el usuario elija la opción de salida (5). Al final del programa, se vacía la pila y se libera la memoria asignada a la pila. Estos son los resultados:

```

-----CONTROL DE PILA-----

1. Mostrar Datos
2. Agregar Datos a la Pila
3. Remover Datos de la Pila
4. Mostrar el dato menor de la Pila
5. Salir

Inserte su opcion: 1

Datos en la Pila:

5. 9
4. 1
3. 2
2. 5
1. 4

-----CONTROL DE PILA-----

1. Mostrar Datos
2. Agregar Datos a la Pila
3. Remover Datos de la Pila
4. Mostrar el dato menor de la Pila
5. Salir

Inserte su opcion: 4
El dato menor de la pila es: 1
-----CONTROL DE PILA-----

```

**Ejercicio 2:** Diseña y escribe un programa que ordene una pila de tal forma que los elementos más pequeños estén en la parte superior. Es posible emplear una pila auxiliar, pero no será posible copiar todos los valores a un arreglo. La pila deberá implementar las operaciones push, pop, isEmpty y peek (visualizar el contenido de la pila).

Incluimos la biblioteca estándar de entrada y salida, la biblioteca para el tipo de dato bool, y la biblioteca que contiene funciones para la asignación de memoria dinámica.

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
```

Define una estructura llamada Nodo, que contiene un entero (valor) y un puntero a otro nodo (NodoInferior).

```
struct Nodo{
    int valor;
    struct Nodo* NodoInferior;
};
```

Define una estructura "Pila" que contiene un puntero a Nodo llamado cima y un entero cursor. Esta estructura representa una pila.

```
typedef struct{
    struct Nodo* cima;
    int cursor;
}Pila;
```

Declaración de las funciones crearPila(), consultarCima(), apilar(), Desapilar(), vaciarPila(), isEmpty(), MostrarPila() y Burble\_sort\_Ascendente(). Estas funciones se definirán más adelante en el código.

```
Pila* crearPila();
void consultarCima(Pila *p);
void apilar(Pila *p, struct Nodo dato);
void Desapilar(Pila *p, struct Nodo *datoExtraido);
void vaciarPila(Pila *p);
bool isEmpty(Pila p);
void MostrarPila(Pila *p);
void Burble_sort_Ascendente(Pila *p);
```

La función `crearPila()` crea una instancia de `Pila`, asigna memoria dinámica para ella, establece `cima` como `NULL` y `cursor` como 0, luego retorna un puntero a la pila creada.

```
Pila* crearPila(){
    Pila* pilaAsignacion=(Pila*) malloc(sizeof(Pila));
    pilaAsignacion->cima=NULL;
    pilaAsignacion->cursor=0;
    return pilaAsignacion;
}
```

La función `consultarCima()` recibe un puntero a `Pila` como parámetro y muestra el valor en la cima de la pila si la pila no está vacía.

```
void consultarCima(Pila *p){
    if(!isEmpty(*p)){
        printf("Valor en la cima: %d\n", p->cima->valor);
    }
}
```

La función `vaciarPila()` recibe un puntero a `Pila` y desapila todos los elementos de la pila mediante llamadas a la función `Desapilar()`.

```
void vaciarPila(Pila *p){
    while(!isEmpty(*p)){
        struct Nodo datoExtraido;
        Desapilar(p,&datoExtraido);
    }
}
```

La función `apilar()` recibe un puntero a `Pila` y un `Nodo` como parámetros. Crea un nuevo nodo, asigna el valor proporcionado y establece el nodo inferior como la cima actual de la pila. Luego actualiza la cima de la pila con el nuevo nodo y aumenta el contador del cursor.

```
void apilar(Pila *p, struct Nodo dato){
    struct Nodo* nodoUsuario=(struct Nodo*) malloc(sizeof(struct Nodo));
    nodoUsuario->valor=dato.valor;
    nodoUsuario->NodoInferior=p->cima;
    p->cima=nodoUsuario;
    p->cursor++;
}
```

La función `Desapilar()` recibe un puntero a `Pila` y un puntero a `Nodo` como parámetros. Extrae el valor de la cima de la pila, actualiza la cima y el cursor, y libera la memoria del nodo desapilado.

```
void Desapilar(Pila *p, struct Nodo *datoExtraido){
    struct Nodo* aux=p->cima;
    datoExtraido->valor=aux->valor;
    p->cima=aux->NodoInferior;//p->cima=p->cima->NodoInferior
    aux->NodoInferior=NULL;
    p->cursor--;
    free(aux);
}
```

La función isEmpty() recibe una pila como parámetro y verifica si la cima de la pila es NULL. Si es así, la pila está vacía y devuelve true; de lo contrario, devuelve false.

```
bool isEmpty(Pila p){
    if(p.cima==NULL){
        return true;
    } else{
        return false;
    }
}
```

La función MostrarPila() recibe un puntero a Pila y muestra los datos almacenados en la pila iterando a través de los nodos de la pila, mostrando su valor y disminuyendo la cantidad mientras avanza hacia el nodo inferior. Si la pila está vacía, muestra un mensaje indicando que no hay datos.

```
void MostrarPila(Pila* p){
    printf ("\nDatos en la Pila:\n\n");
    struct Nodo *aux;
    aux = p->cima; //Se trabaja con una copia de la Pila "Principal"
    if (aux == NULL) {
        printf ("No hay Datos\n");
    }
    int cantidad = p->cursor;
    while (aux) {
        printf ("%d. %d\n", cantidad, aux->valor);
        aux = aux->NodoInferior;
        cantidad--;
    }
    free(aux);
}
```

La función Burble\_sort\_Ascendente() recibe un puntero a Pila y ordena los datos de la pila de forma ascendente utilizando el algoritmo de ordenamiento Bubble Sort.

```
void Burbble_sort_Ascendente(Pila *p){
    struct Nodo *aux;
    aux = p->cima; //Se trabaja con una copia de la Pila "Principal"
    if (aux == NULL) {
        printf ("No hay Datos\n");
    }else{
        while(aux!=NULL){
            struct Nodo *aux2 = aux->NodoInferior;
            while(aux2!=NULL){
                if(aux->valor>aux2->valor){
                    int aux3 = aux2->valor;
                    aux2->valor = aux->valor;
                    aux->valor=aux3;
                }
                aux2=aux2->NodoInferior;
            }
            aux=aux->NodoInferior;
        }
    }
}
```

La función main() es la función principal del programa. Declara variables locales, incluyendo un puntero a Pila llamado p y un Nodo llamado dato. Luego muestra un menú de opciones y permite al usuario interactuar con la pila utilizando las funciones definidas anteriormente. El bucle do-while se ejecuta hasta que el usuario elige la opción "6" para salir. Después de salir del bucle, se vacía la pila, se libera la memoria asignada y retorna 0 para finalizar el programa. Estos son los resultados:



```
Datos en la Pila:
5. 1
4. 4
3. 5
2. 2
1. 3

-----CONTROL DE PILA-----

1. Mostrar Datos
2. Mostrar el valor en la cima de la Pila
3. Agregar Datos a la Pila
4. Remover Datos de la Pila
5. Ordenar la pila con los datos menores hasta la cima
6. Salir

Inserte su opcion: 5

Sus datos estan ordenados!

-----CONTROL DE PILA-----

1. Mostrar Datos
2. Mostrar el valor en la cima de la Pila
3. Agregar Datos a la Pila
4. Remover Datos de la Pila
5. Ordenar la pila con los datos menores hasta la cima
6. Salir

Inserte su opcion: 1

Datos en la Pila:
5. 1
4. 2
3. 3
2. 4
1. 5
```

**Ejercicio 3:** Diseña y escribe un programa que, empleando Orientación a Objetos, determine cuál es el par más cercano de puntos en 3D. Los puntos tendrán la estructura. Preferentemente emplea la distancia Euclidiana. Deberás probar con al menos 20 puntos los cuales podrán generarse de manera aleatoria, capturados a mano o leídos desde un archivo.

El código define una clase llamada Punto3D, que representa un punto en el espacio 3D. Tiene atributos x, y, z para almacenar las coordenadas del punto. La clase también tiene un método distancia que calcula la distancia entre dos puntos

utilizando la fórmula de distancia euclidiana.

```
import math
import random

class Punto3D:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def distancia(self, otro_punto):
        dx = self.x - otro_punto.x
        dy = self.y - otro_punto.y
        dz = self.z - otro_punto.z
        return math.sqrt(dx**2 + dy**2 + dz**2)
```

La función encontrar\_par\_mas\_cercano recibe una lista de puntos como entrada y recorre todos los pares posibles de puntos para encontrar el par con la distancia más pequeña entre ellos. Inicializa las variables par\_cercano y distancia\_minima para realizar un seguimiento del par más cercano y su distancia. La distancia entre cada par de puntos se calcula utilizando el método distancia de la clase Punto3D. La función devuelve el par de puntos más cercanos.

```
def encontrar_par_mas_cercano(puntos):
    par_cercano = (None, None)
    distancia_minima = float('inf')

    for i in range(len(puntos)):
        for j in range(i + 1, len(puntos)):
            distancia = puntos[i].distancia(puntos[j])
            if distancia < distancia_minima:
                distancia_minima = distancia
                par_cercano = (puntos[i], puntos[j])

    return par_cercano
```

La función generar\_puntos\_aleatorios genera una cantidad específica de puntos aleatorios en 3D dentro de un rango (-10 a 10) para cada coordenada (x, y, z). La función crea instancias de la clase Punto3D para cada punto aleatorio, imprime los puntos generados y devuelve una lista de los puntos.

```
def generar_puntos_aleatorios(n):
    puntos = []
    for _ in range(n):
        x = round(random.uniform(-10, 10), 3)
        y = round(random.uniform(-10, 10), 3)
        z = round(random.uniform(-10, 10), 3)
        punto = Punto3D(x, y, z)
        puntos.append(punto)
        print("Punto generado: P({}, {}, {})".format(punto.x, punto.y, punto.z))
    return puntos
```

La función `ingresar_puntos_manualmente` permite al usuario ingresar manualmente las coordenadas de cada punto. Solicita al usuario la cantidad de puntos y luego itera esa cantidad de veces para recopilar las coordenadas. Al igual que `generar_puntos_aleatorios`, crea instancias de `Punto3D`, imprime los puntos ingresados y devuelve una lista de los puntos.

```
def ingresar_puntos_manualmente(n):
    puntos = []
    for i in range(n):
        print("Punto #{}".format(i + 1))
        x = float(input("x: "))
        y = float(input("y: "))
        z = float(input("z: "))
        punto = Punto3D(x, y, z)
        puntos.append(punto)
        print("Punto ingresado: P({}, {}, {})".format(punto.x, punto.y, punto.z))
    return puntos
```

La parte principal del código le pide al usuario que seleccione cómo desea generar los puntos (aleatoriamente, manualmente o desde un archivo) y maneja cada caso en consecuencia. Si el usuario elige leer puntos desde un archivo, el código lee los puntos del archivo y crea instancias de `Punto3D` para cada línea del archivo. Si el archivo no se encuentra, se generan puntos aleatorios en su lugar. En ambos casos, los puntos se almacenan en la lista `puntos`.

```

print("¿Cómo desea generar los puntos?")
print("1. Aleatoriamente")
print("2. Manualmente")
print("3. Desde un archivo")
opcion = int(input("Opción: "))
while opcion != 1 and opcion != 2 and opcion != 3:
    print("Opción inválida.")
    opcion = int(input("Opción: "))
print()

if opcion == 1:
    puntos = generar_puntos_aleatorios(20)
elif opcion == 2:
    puntos = ingresar_puntos_manualmente(20)
elif opcion == 3:
    print("Si el archivo esta en la misma carpeta que este programa, solo ingrese el nombre de")
    print("Si el archivo esta en otra carpeta, ingrese la ruta completa del archivo.")
    archivo = input("Ingrese el nombre del archivo de puntos: ")
    puntos = []
    try:
        print("Leyendo puntos desde el archivo {}.format(archivo))
        with open(archivo, 'r') as file:
            for linea in file:
                valores = linea.strip().strip('P()').split(',')
                x = round(float(valores[0]), 3)
                y = round(float(valores[1]), 3)
                z = round(float(valores[2]), 3)
                punto = Punto3D(x, y, z)
                puntos.append(punto)
                print("Punto leído: P({}, {}, {})".format(punto.x, punto.y, punto.z))
    except FileNotFoundError:
        print("Archivo no encontrado. Se generarán puntos aleatorios en su lugar.")
        puntos = generar_puntos_aleatorios(20)

```

Después de obtener los puntos, el código llama a la función encontrar\_par\_mas\_cercano para encontrar el par de puntos más cercanos.

```

# Encontrar el par más cercano
par_cercano = encontrar_par_mas_cercano(puntos)

```

Por último, el código imprime el resultado mostrando las coordenadas del par más cercano de puntos y su distancia. Si no se encuentra ningún par cercano, muestra un mensaje que lo indica.

```

# Imprimir el resultado
if par_cercano[0] is not None and par_cercano[1] is not None:
    print("El par más cercano de puntos en 3D es:")
    print("P1: P({}, {}, {})".format(par_cercano[0].x, par_cercano[0].y, par_cercano[0].z))
    print("P2: P({}, {}, {})".format(par_cercano[1].x, par_cercano[1].y, par_cercano[1].z))
    print("Distancia:", par_cercano[0].distancia(par_cercano[1]))
else:
    print("No se encontró un par de puntos cercanos.")

```

Estos son los resultados:

```
¿Cómo desea generar los puntos?
1. Aleatoriamente
2. Manualmente
3. Desde un archivo
Opción: 1

Punto generado: P(8.703, 9.452, -8.838)
Punto generado: P(-0.914, 6.699, -2.982)
Punto generado: P(3.52, 2.523, 5.177)
Punto generado: P(2.409, -1.7, -2.972)
Punto generado: P(8.486, -0.686, 3.58)
Punto generado: P(-0.359, -7.698, -2.885)
Punto generado: P(1.456, -5.735, -4.212)
Punto generado: P(8.261, -6.93, -2.424)
Punto generado: P(-1.422, 1.491, 3.209)
Punto generado: P(2.64, -1.227, 9.04)
Punto generado: P(7.905, 6.667, 6.2)
Punto generado: P(2.853, -3.613, 7.078)
Punto generado: P(7.726, -0.458, -0.426)
Punto generado: P(-8.423, -1.09, -6.224)
Punto generado: P(5.246, 1.446, -4.631)
Punto generado: P(3.639, 2.616, -9.935)
Punto generado: P(5.415, -5.94, 3.433)
Punto generado: P(-0.171, 7.454, 3.785)
Punto generado: P(6.701, 3.857, 5.396)
Punto generado: P(-0.298, 7.024, -2.361)
El par más cercano de puntos en 3D es:
P1: P(-0.914, 6.699, -2.982)
P2: P(-0.298, 7.024, -2.361)
Distancia: 0.9331248576691118
```

**Ejercicio 4:** A partir del ejemplo revisado en el archivo family.pl, amplía la base de conocimientos de tal manera que se pueden agregar reglas para los casos de bisabuelo, tatarabuelo y sobrino.

```
% Facts: parent(Child, Parent)
```

```
parent(john, mary).
```

```
parent(john, mike).
```

```
parent(mary, alice).
```

```
parent(mary, george).
```

```
parent(mike, lisa).
```

parent(mike, james).

parent(susan, alice).

parent(susan, george).

parent(lisa, carol).

parent(lisa, paul).

parent(james, karen).

parent(james, daniel).

% Rule: grandparent(Grandchild, Grandparent)

grandparent(Grandchild, Grandparent) :-

parent(Grandchild, Parent),

parent(Parent, Grandparent).

% Rule: sibling(Person1, Person2)

sibling(Person1, Person2) :-

parent(Person1, Parent),

parent(Person2, Parent),

Person1 \= Person2.

% Rule: cousin(Person1, Person2)

cousin(Person1, Person2) :-

parent(Person1, Parent1),

parent(Person2, Parent2),

sibling(Parent1, Parent2).

% Rule: bisabuelo(Bisabuelo, Nieto)

bisabuelo(Bisabuelo, Nieto) :-

parent(Nieto, Padre),

```
parent(Padre, Abuelo),
parent(Abuelo, Bisabuelo).
```

% Rule: tatarabuelo(Tatarabuelo, Descendiente)

tatarabuelo(Tatarabuelo, Descendiente) :-

```
parent(Descendiente, Hijx),
parent(Hijx, Nietx),
parent(Nietx, Bisnietx),
parent(Bisnietx, Tatarabuelo).
```

% Rule: sobrino(Sobrino, TioTia)

sobrino(Sobrino, TioTia) :-

```
sibling(Sobrino, PadreOMadre),
parent(PadreOMadre, TioTia).
```

Los primeros hechos establecidos definen las relaciones parentales entre varias personas. Por ejemplo, se establece que "john" es padre de "mary" y "mike", "mary" es madre de "alice" y "george", etc. Estos hechos proporcionan la base para establecer las relaciones familiares en el programa.

- La regla Grandparent: Esta regla define el predicado grandparent/2. Toma dos argumentos: Grandchild (nieta/nieto) y Grandparent (abuelo/abuela).

La regla establece que Grandparent es el abuelo o abuela de Grandchild si hay un Parent (padre/madre) en común entre ellos. Es decir, si Grandchild tiene un Parent y ese Parent tiene un Grandparent, entonces Grandparent es un abuelo o abuela de Grandchild.

- La regla sibling define el predicado sibling/2. Toma dos argumentos: Person1 y Person2. La regla establece que Person1 y Person2 son hermanos si tienen el mismo padre/madre (Parent), pero no son la misma persona (Person1 \= Person2).

- La regla cousin define el predicado cousin/2. Toma dos argumentos: Person1 y Person2. La regla establece que Person1 y Person2 son primos si tienen padres que son hermanos (sibling(Parent1, Parent2)). Es decir, si los padres de Person1 y Person2 son hermanos, entonces Person1 y Person2 son primos.
- La regla bisabuelo define el predicado bisabuelo/2. Toma dos argumentos: Bisabuelo y Nieto. La regla establece que Bisabuelo es bisabuelo de Nieto si existe una secuencia de relaciones de parentesco que va desde Nieto hasta Bisabuelo. Para ello, se utilizan los predicados parent/2 para establecer las relaciones entre Nieto, Padre, Padre, Abuelo, y Abuelo, Bisabuelo. Es decir, si Nieto es hijo de Padre, Padre es hijo de Abuelo, y Abuelo es hijo de Bisabuelo, entonces Bisabuelo es bisabuelo de Nieto.
- La regla tatarabuelo define el predicado tatarabuelo/2. Toma dos argumentos: Tatarabuelo y Descendiente. La regla establece que Tatarabuelo es tatarabuelo de Descendiente si hay una secuencia de relaciones de parentesco que va desde Descendiente hasta Tatarabuelo a través de tres niveles de ancestros. Utiliza los predicados parent/2 para establecer las relaciones entre Descendiente, Hijo, Hijo, Nieto, Nieto, Bisnieto, y Bisnieto, Tatarabuelo.
- La regla sobrino define el predicado sobrino/2. Toma dos argumentos: Sobrino y TioTia. La regla establece que Sobrino es sobrino de TioTia si comparten un padre o madre (PadreOMadre) y TioTia es el padre o madre de PadreOMadre. Utiliza el predicado sibling/2 para establecer que Sobrino comparte un padre o madre con TioTia. Luego, utiliza el predicado parent/2 para establecer la relación entre PadreOMadre y TioTia.

Estos son los resultados:

<pre> SWI-Prolog (AMD64, Multi-threaded, version 9.0.4) File Edit Settings Run Debug Help Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4) SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software. Please run ?- license. for legal details.  For online help and background, visit https://www.swi-prolog.org For built-in help, use ?- help(Topic), or ?- apropos(Word).  ?- consult('C:/Users/Comp/Desktop/P4.pl'). true.  ?- sobrino(john, alice). false.  ?- grandparent(john, alice). true.  ?- sibling(susan, karen). false.  ?- bisabuelo(karen, john). true.  ?- tatarabuelo(karen, john). false.  ?- </pre>	<pre> P4: Bloc de notes Archivo Edición Formato Ver Ayuda  % Facts: parent(Child, Parent) parent(john, mary). parent(john, mike). parent(mary, alice). parent(mary, george). parent(mike, lisa). parent(mike, james). parent(susan, alice). parent(susan, george). parent(lisa, carol). parent(lisa, paul). parent(james, karen). parent(james, daniel).  % Rule: grandparent(Grandchild, Grandparent) grandparent(Grandchild, Grandparent) :-     parent(Grandchild, Parent),     parent(Parent, Grandparent).  % Rule: sibling(Person1, Person2) sibling(Person1, Person2) :-     parent(Person1, Parent),     parent(Person2, Parent),     Person1 \= Person2.  % Rule: cousin(Person1, Person2) cousin(Person1, Person2) :-     parent(Person1, Parent1),     parent(Person2, Parent2), </pre>
--	--



**Ejercicio 5:** Elabora uno o más programas que resuelvan lo siguiente.

- Cálculo de la factorial.
- Secuencia de Fibonacci.

**Cálculo de la factorial:**

```
(defun factorial (n)
```

```
  (if (<= n 1)
```

```
    1
```

```
    (* n (factorial (- n 1)))))
```

```
(defun calcular-factorial ()
```

```
  (format t "Ingrese un número entero para calcular su factorial: ")
```

```
  (let ((numero (parse-integer (read-line)))))
```

```
    (format t "El factorial de a es: ~a%" numero (factorial numero))))
```

```
(calcular-factorial)
```

En las primeras 4 líneas del código:

```
(defun factorial (n)
```

```
  (if (<= n 1)
```

```
    1
```

```
    (* n (factorial (- n 1)))))
```

Se define la función factorial, que toma un argumento n. La función calcula la factorial de n utilizando una estructura de control condicional.

Si n es menor o igual a 1, devuelve 1. De lo contrario, calcula la factorial multiplicando n por el factorial de n-1.

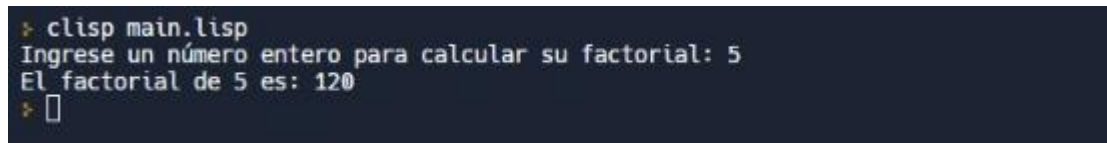
De las líneas 5 a la 8:

```
(defun calcular-factorial ()
  (format t "Ingrese un número entero para calcular su factorial: ")
  (let ((numero (parse-integer (read-line))))
    (format t "El factorial de a es: ~a%" numero (factorial numero)))))
```

Estas líneas definen la función calcular-factorial. No toma argumentos. La función solicita al usuario que ingrese un número entero para calcular su factorial.

Luego, utiliza la función read-line para leer la entrada del usuario y parse-integer para convertirlo en un número entero. A continuación, utiliza format para imprimir el resultado de la factorial utilizando el valor ingresado y la función factorial.

La última línea de código únicamente llama a ejecutar la función para realizar el programa.



```
* clisp main.lisp
Ingrese un número entero para calcular su factorial: 5
El factorial de 5 es: 120
* □
```

### Secuencia de Fibonacci:

```
(defun fibonacci (n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (t (+ (fibonacci (- n 1)) (fibonacci (- n 2))))))
```

```
(defun calcular-fibonacci ()
  (format t "Ingrese un número entero para calcular la serie de Fibonacci: ")
  (let ((numero (parse-integer (read-line))))
    (format t "Serie de Fibonacci para a:%" numero)
    (dotimes (i numero)
      (format t "Fib(a) = ~a%" i (fibonacci i)))))
```

```
(format t "Resultado final: a%" (fibonacci numero))))
```

```
(calcular-fibonacci)
```

Las primeras 4 líneas de código:

```
(defun fibonacci (n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (t (+ (fibonacci (- n 1)) (fibonacci (- n 2))))))
```

En estas primeras líneas se define la función fibonacci, que toma un argumento n. La función utiliza una estructura de control condicional cond para calcular el n-ésimo número de la serie de Fibonacci. Si n es igual a 0, devuelve 0. Si n es igual a 1, devuelve 1. Para cualquier otro valor de n, utiliza la recursividad para calcular el número sumando los dos números anteriores de la serie de Fibonacci: (fibonacci (- n 1)) y (fibonacci (- n 2)).

De las líneas de código 5 a la 11:

```
(defun calcular-fibonacci ()
  (format t "Ingrese un número entero para calcular la serie de Fibonacci: ")
  (let ((numero (parse-integer (read-line))))
    (format t "Serie de Fibonacci para a:%" numero)
    (dotimes (i numero)
      (format t "Fib(a) = ~a%" i (fibonacci i)))
    (format t "Resultado final: a%" (fibonacci numero))))
```

En todas estas líneas de código que son el cuerpo del programa se define la función calcular-fibonacci. No toma argumentos. La función solicita al usuario que ingrese un número entero para calcular la serie de Fibonacci. Luego, utiliza parse-integer y read-line para leer y convertir la entrada del usuario en un número entero.

A continuación, utiliza format para imprimir la etiqueta "Serie de Fibonacci para a:" seguida del número ingresado. La función dotimes se utiliza para iterar desde 0 hasta numero - 1 (el número ingresado por el usuario) utilizando la variable i como contador. En cada iteración, se utiliza format para imprimir el número de Fibonacci correspondiente a i utilizando la función fibonacci. Después de imprimir la serie de Fibonacci para cada número de la secuencia, se utiliza format para imprimir el resultado final, que es el número de Fibonacci correspondiente al número ingresado por el usuario.

Nuevamente en la última línea de código (12), se manda a llamar la función para ejecutar el programa

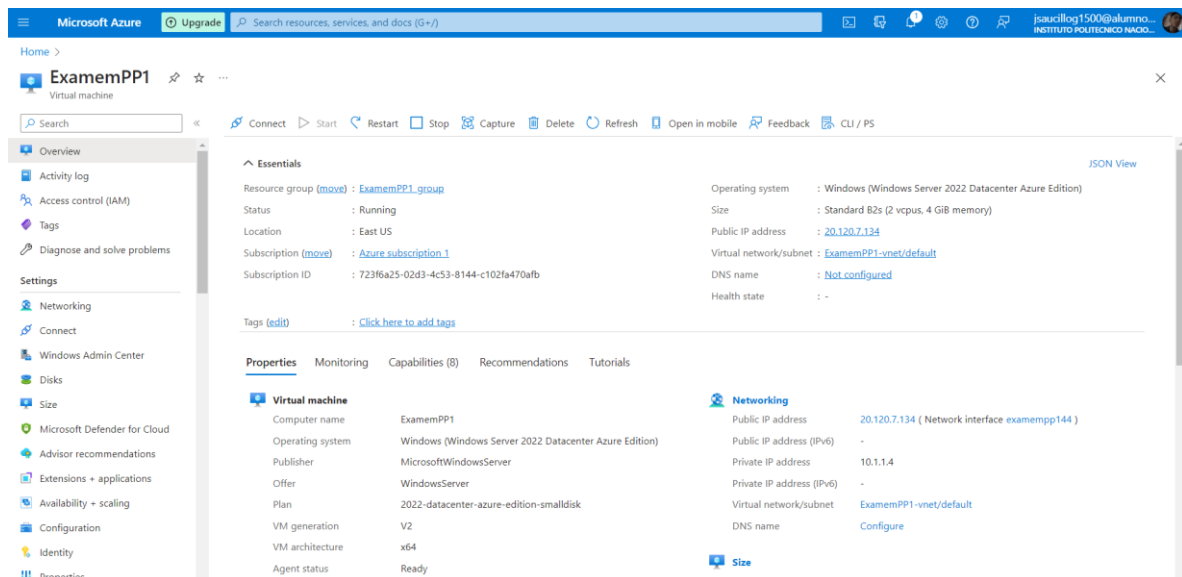
```

❖ clisp main.lisp
Ingrese un número entero para calcular la serie de Fibonacci: 5
Serie de Fibonacci para 5:
Fib(0) = 0
Fib(1) = 1
Fib(2) = 1
Fib(3) = 2
Fib(4) = 3
Resultado final: 5
❖ 

```

## PARTE II

Se opto por crear una maquina virtual en la nube directamente en Microsoft Azure.



Usamos el plan estudiantil que nos da 200\$ para crear maquinas virtuales. Creamos la máquina virtual con Windows (Windows Server 2022 Datacenter Azure Edition) y cuyo nombre es Examen PP1.

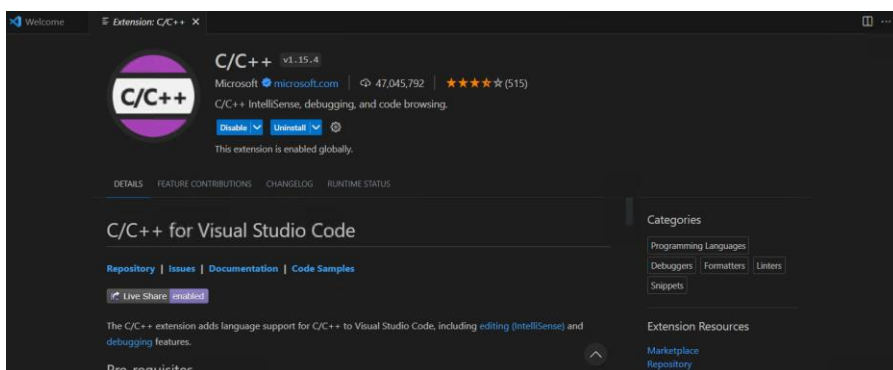
Para conectarse simplemente se utiliza:



Usuario:Fixx910

Contraseña: F81Jfdoh22304

Una vez dentro de la maquina virtual se instalo Visual Studio Code junto con las extensiones y compiladores de C y Phyton.



Ejecución de los codigos en la maquina virtual:

```

142
143
144     case 4:
145         puts("Cuantos valores quiere quitar a la pila?");
146         fflush(stdin);
147         scanf("%d",&cantidadDatos);
148         for(int i=0; i<cantidadDatos; i++){
149             Desapilar(p,&dato);
150             printf("Valor extraido: %d\n",dato.valor);
151         }
152         break;
153
154     case 5:
155         Burbule_sort_Ascendente(p);
156         puts("\nSus datos estan ordenados!");
157
158     default:
159         break;
160 }while (opc!=6);
161 vaciarPila(p);
162 free(p);
163 return 0;
164 }

```

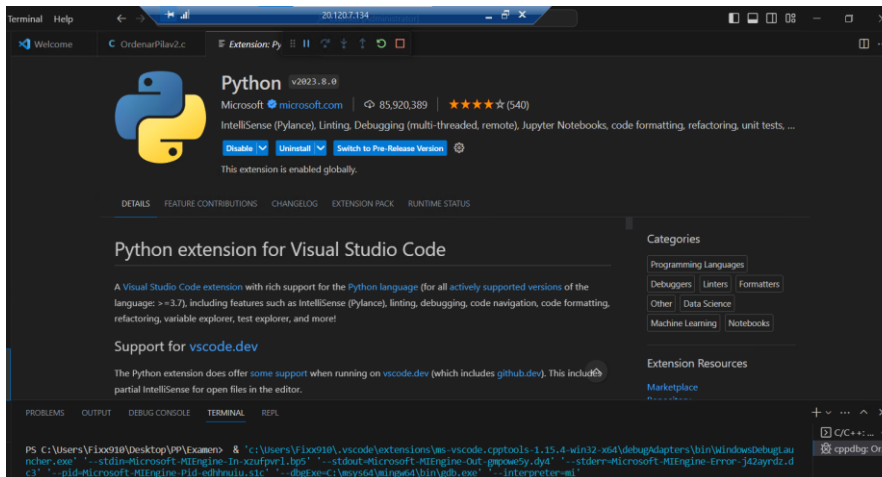
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL REPL

```

PS C:\Users\Flores\OneDrive\Desktop\PP\Examen> & 'c:\Users\Flores\OneDrive\Desktop\PP\Examen\OrdenarPila2.exe'
-----CONTROL DE PILA-----
1. Mostrar Datos
2. Agregar el valor en la cima de la Pila
3. Remover Datos de la Pila
4. Ordenar la pila con los datos menores hasta la cima
5. Salir

```

Python:



Ejeci3n de c3digo pyhton en la maquina virtual:

```

34
35 def generar_puntos_aleatorios(n):
36     puntos = []
37     for _ in range(n):
38         x = round(random.uniform(-10, 10), 3)
39         y = round(random.uniform(-10, 10), 3)
40         z = round(random.uniform(-10, 10), 3)
41         punto = Punto3D(x, y, z)
42         puntos.append(punto)
43         print("Punto generado: P({}, {}, {})".format(punto.x, punto.y, punto.z))
44     return puntos
45
46 def ingresar_puntos_manualmente(n):
47     puntos = []
48     for i in range(n):
49         print("Punto #{}".format(i + 1))
50         x = float(input("x: "))
51         y = float(input("y: "))
52         z = float(input("z: "))
53         punto = Punto3D(x, y, z)
54         puntos.append(punto)
55         print("Punto ingresado: P({}, {}, {})".format(punto.x, punto.y, punto.z))
56     return puntos
57

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL REPL


PS C:\Users\Fixx910\Desktop\PP\Examen> & C:/Users/Fixx910/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Fixx910/Desktop/PP/Examen/Ejer  
cicio3.py  
¿Cómo desea generar los puntos?  
1. Aleatoriamente  
2. Manualmente  
3. Desde un archivo  
Opción: 3

Si el archivo esta en la misma carpeta que este programa, solo ingrese el nombre del archivo.  
Si el archivo esta en otra carpeta, ingrese la ruta completa del archivo.  
Ingrese el nombre del archivo de puntos: puntos.txt


## Instalación necesaria para prolog

← ↻ 🔒 <https://www.swi-prolog.org/download/stable/...> 🔍 ⚙️ 👤 ⋮

**News: SWEEP: A SWI-Prolog mode for GNU-Emacs based on semantic analysis by Prolog** Search Documentation:

 **Download binary**

**HOME** **DOWNLOAD** DOCUMENTATION TUTORIALS COMMUNITY

 Windows antivirus software works using *signatures* and *heuristics*. Using the huge amount of viruses and m  
arbitrary executables are often *falsily classified as malicious*. [Google Safe Browsing](#), used by most modern b  
classifies our Windows binaries as malware. You can use e.g., [virustotal](#) to verify files with a large number of anti

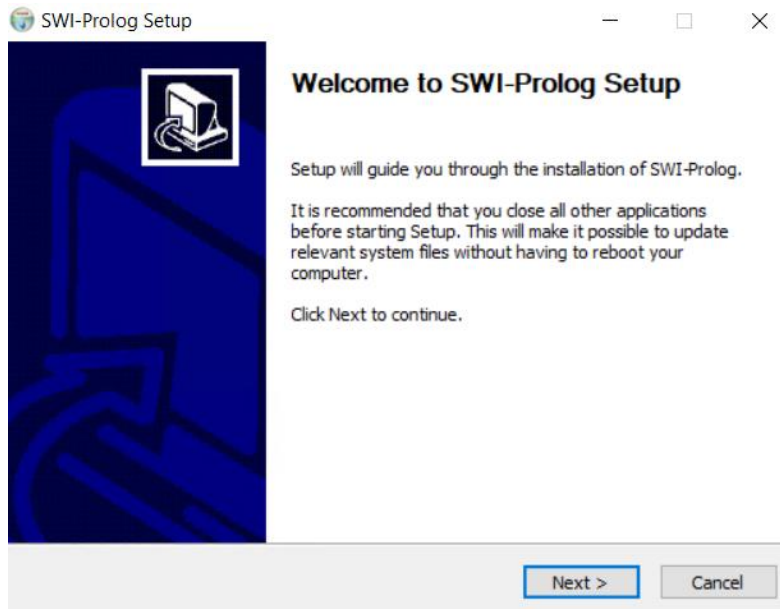
Our Windows binaries are cross-compiled on an isolated Linux container. The integrity of the binaries on the ser  
by validating its SHA256 fingerprint.

Please select the checkbox below to enable the actual download link.

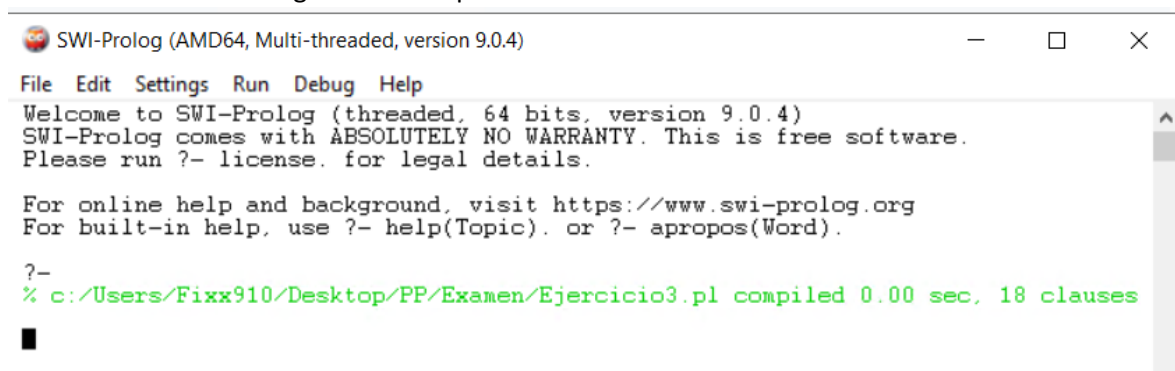
☒ I understand

[Download swipl-9.0.4-1.x64.exe](#) (SHA256: 33758f1c2dd190df9c8828d2dcb39166ad10d31d78f1198812e6d0f33b71c73b)

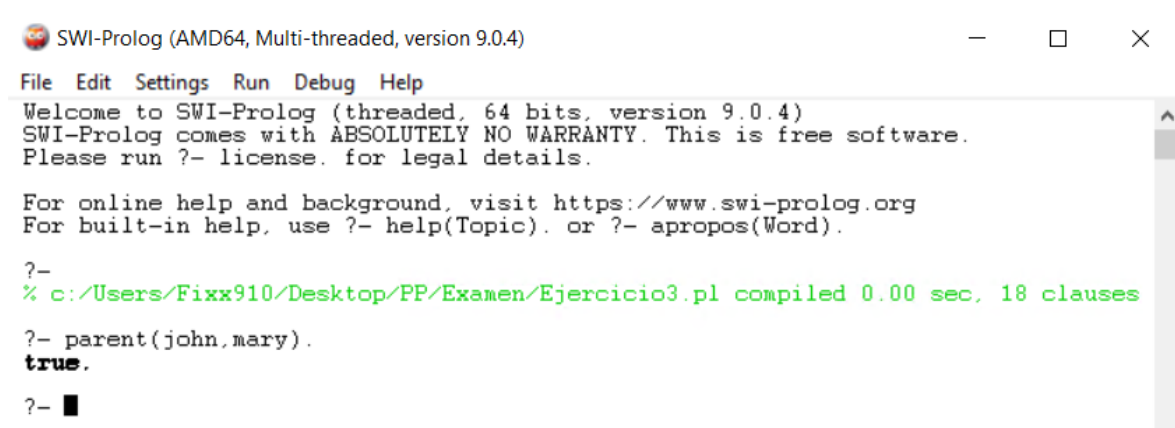
[VIRUSTOTAL Scan Result](#)



Una vez instalado se carga el archivo .pl

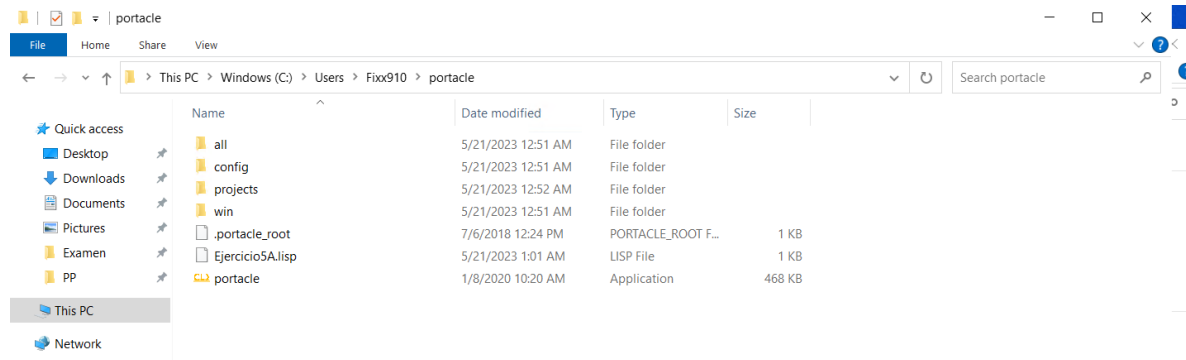


Demostración de que efectivamente funcionan las consultas en la maquina virtual:

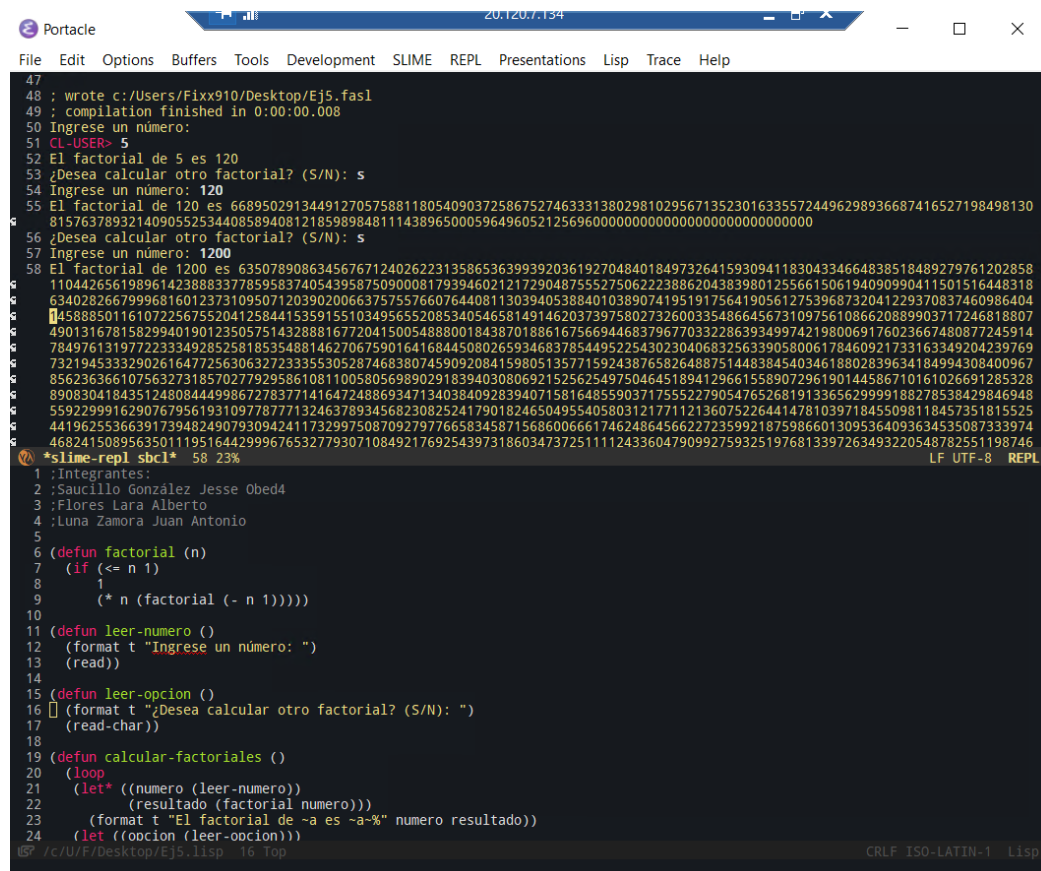


Para el ambiente de Lisp lo más sencillo fue Portacle que es un ambiente totalmente portable de common Lisp.





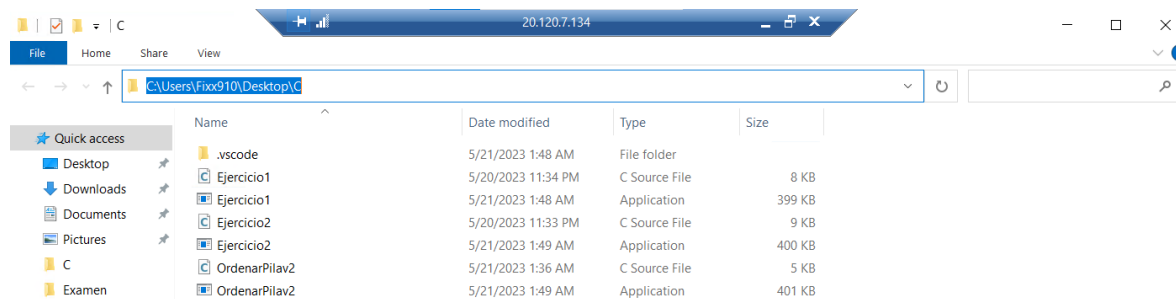
Se ejecuta y dentro podemos compilar los archivos .lisp:



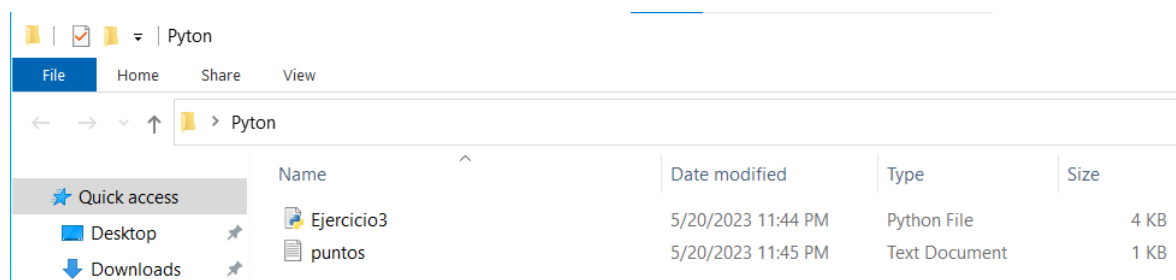
A continuación se muestran los codigos y sus respectivas herramientas para comprobar su funcionamiento:



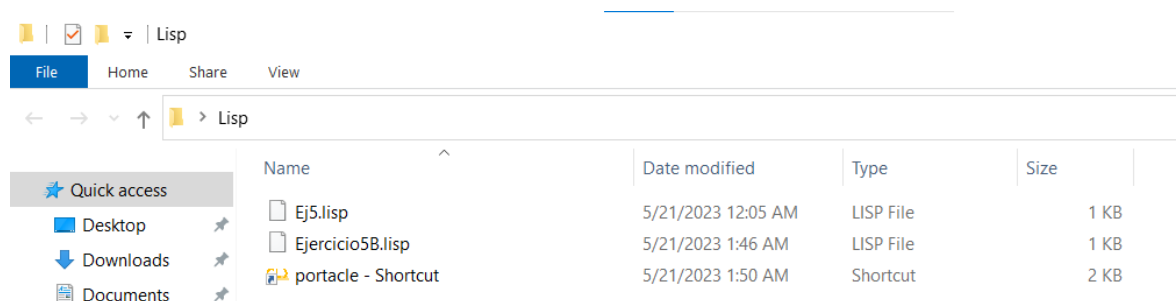
C:



Phyton:



Lisp:



Prolog:

