

MANEJO FUNCIONES EN EL LENGUAJE C++

Elaborado por: Juan Miguel Guanira Erazo

PUCP

REPASO DE CONCEPTOS BÁSICOS:

**Uso de la memoria al
ejecutar un programa.**

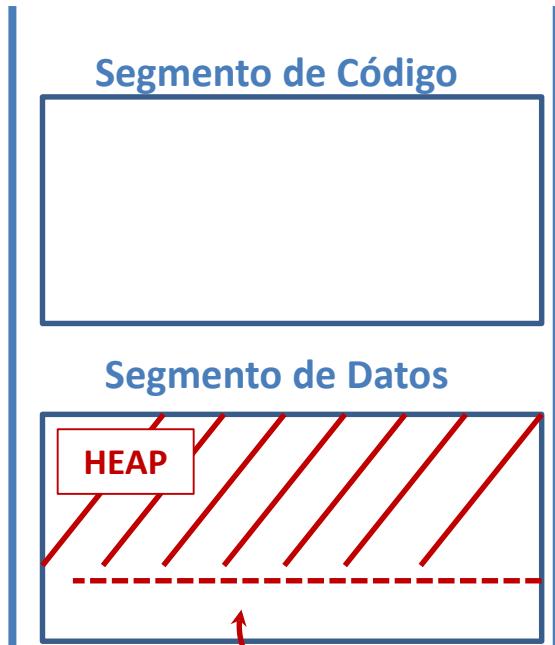
Memoria principal del computador

Programa

```
int main(){
    int a=3, b =75, c;
    c = f(a,b);
    ...
    ...
}
```

```
int f(int x, int y){
    int t;
    t = x + y;
    return t;
}
```

Memoria principal del computador



Programa

```
int main(){
    int a=3, b =75, c;
    c = f(a,b);
    ...
}
```

```
int f(int x, int y){
    int t;
    t = x + y;
    return t;
}
```

Memoria principal del computador

Segmento de Código

Segmento de Datos

Programa

```
int main(){
    int a=3, b =75, c;
    c = f(a,b);
    ...
}
```

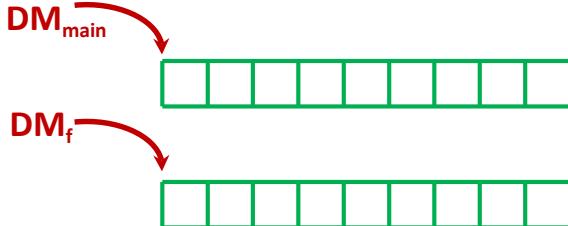
```
int f(int x, int y){
    int t;
    t = x + y;
    return t;
}
```

Tabla de identificadores

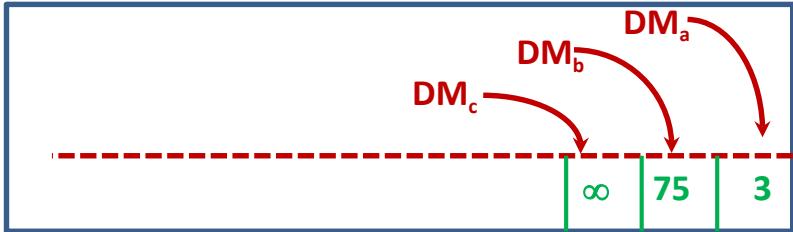
Nombre	Dirección de memoria
main	DM _{main}
f	DM _f
a	DM _a
b	DM _b
c	DM _c

Memoria principal del computador

Segmento de Código



Segmento de Datos



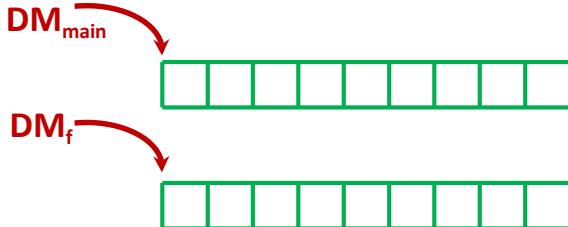
Programa

```
int main(){  
    int a=3, b =75, c;  
    c = f(a,b);  
    ...  
    ...  
}
```

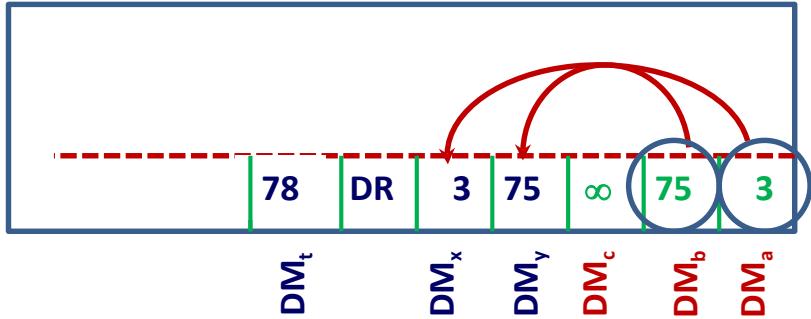
```
int f(int x, int y){  
    int t;  
    t = x + y;  
    return t;  
}
```

Memoria principal del computador

Segmento de Código

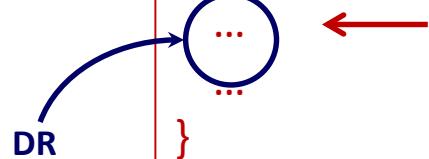


Segmento de Datos



Programa

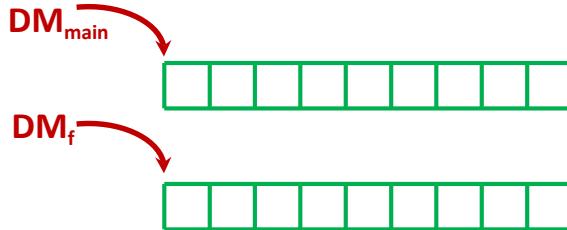
```
int main(){
    int a=3, b =75, c;
    c = f(a,b);
```



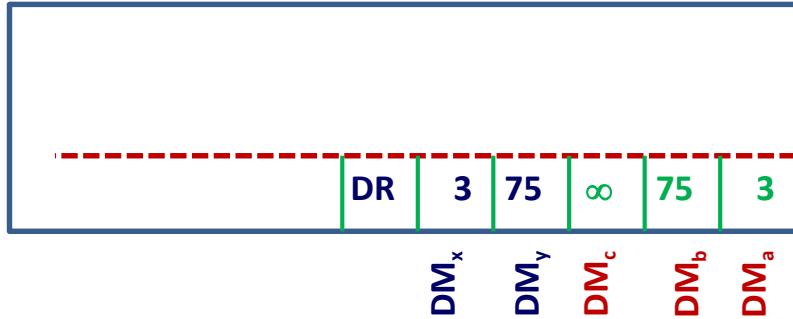
```
int f(int x, int y){
    int t;
    t = x + y;
    return t;
}
```

Memoria principal del computador

Segmento de Código



Segmento de Datos



DR →

Programa
int main(){
 int a=3, b =75, c;
 c = f(a,b);

...
...
}

int f(int x, int y){
 int t;
 t = x + y;
 return t;
}

Arreglos y punteros

```
int arr[5];    arr[0] ... arr[4]
```

```
int arr[5] = {1,2,3,4,5};
```

```
int arr[5] {1,2,3,4,5};
```

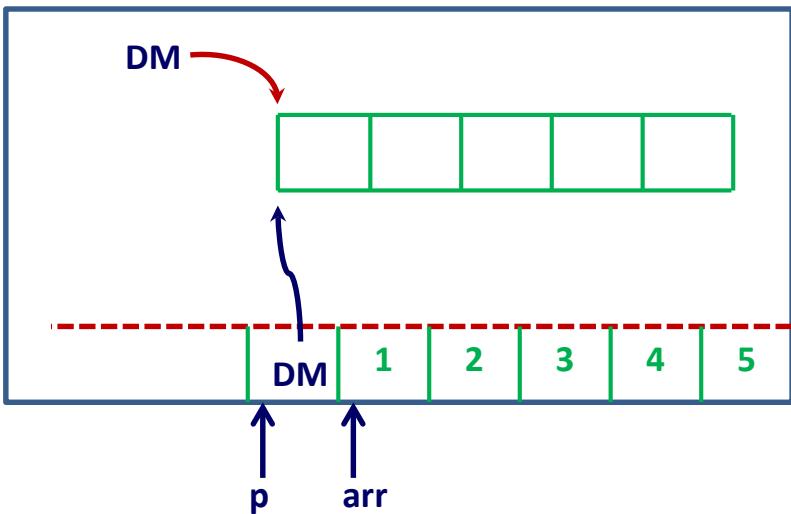
```
int *p;
```

```
p = new int[5];
```

```
p = new int[5] {1,2,3,4,5};
```

Memoria principal del computador

Segmento de Datos

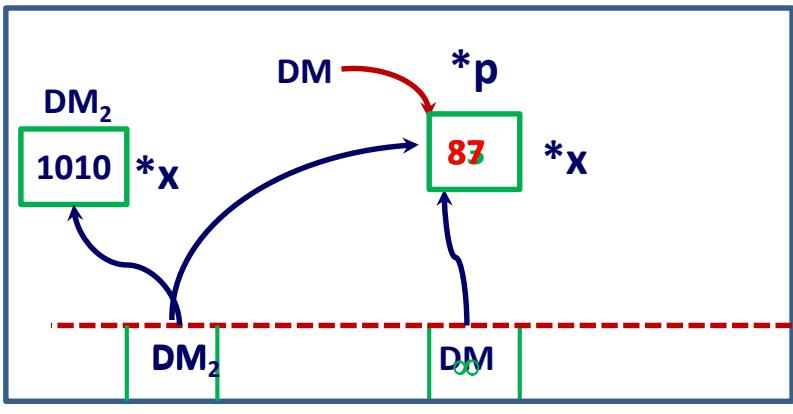


```
int arr[5] = {1,2,3,4,5};  
int *p;
```

```
p = new int[5];
```

Memoria principal del computador

Segmento de Datos



```
int *p;
```

```
p = new int;
```

```
*p = 33;
```

```
porValor(p);
```

```
cout<<"P= "<<*p<<endl;
```

```
void porValor(int *x){
```

```
*x = 87;
```

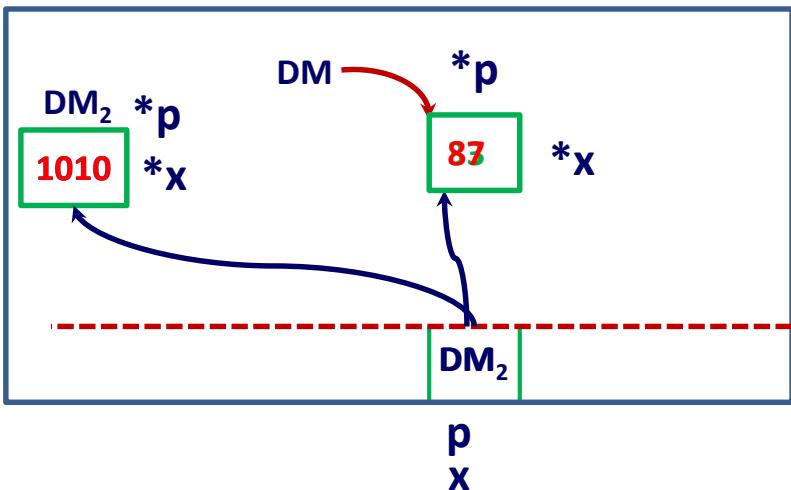
```
x = new int;
```

```
*x = 1010;
```

```
}
```

Memoria principal del computador

Segmento de Datos



```
int *p;  
p = new int;  
*p = 33;
```

```
porRef(p);
```

```
cout<<"P= "<<*p<<endl;
```

```
void porRef(int *&x){
```

```
*x = 87;
```

```
x = new int;
```

```
*x = 1010;
```

```
}
```

Parámetros de una función

Parámetros por valor

Parámetros por referencia

Invocación: `f (a);`

`f (a);`

Encabezado: `T f (int);`

`T f (int &);`

Implemen-
tación:

`T f (int a){`

`T f (int &a){`

Parámetros con valores por defecto

Encabezado: `T f (int = 10, int = 7);`

Implementación: `T f (int a, int b){`

Invocación:

- `f (a, b);`
- `f (a);`
- `f ();`

Sobrecarga de funciones

Supongamos que hemos definido las siguientes funciones en un mismo proyecto:

```
int f (int a){  
    return a*10;  
}
```

```
int f (int a){  
    return a+5;  
}
```

¿Qué pasaría si en algún punto del programa escribimos:

b = f (9);

**!ERROR por
AMBIGÜEDAD!**

Pero si ahora definimos estas otras funciones en un mismo proyecto:

```
int f (int a){  
    return a*10;  
}
```

```
int f (int a, int b){  
    return a+5;  
}
```

¿Qué pasaría si en algún punto del programa escribimos:

b = f (9,5);

**Se ejecuta la
segunda función**

b = f (9);

**Se ejecuta la
primera función**

Lo mismo pasará si definimos:

```
int f (int a){
```

...

```
int f (int *a){
```

...

```
int f (struct St a){
```

...

```
int f (class Cl a){
```

...

DEFINICIÓN:

La sobrecarga es una propiedad del Lenguaje C++ que permite definir dos o más funciones con el mismo nombre:

REQUISITOS:

Los parámetros deben ser diferentes, ya sea en número o en tipo de dato.

Sobrecarga de operadores

El Lenguaje C++ interpreta toda expresión de la forma:

Operando1 operador Operando2

por ejemplo: a + b

De la siguiente forma:

Operador (Operando1, Operando2)

Es decir, los maneja como si fueran funciones

Se pueden sobrecargar:

++	--	+	-	*	/	%	
&	^	~	>>	<<	==	!=	>
>=	<=	not	and	or	=	+=	-=
*=	/=	%=	=	&=	*=	^=	>>=
<<=	[]	()	->	new	delete		

No se pueden sobrecargar:

::	.	? :
----	---	-----

Restricciones:

- No se pueden crear nuevos operadores, solo se pueden sobrecargar los operadores antes mencionados. Por ejemplo, no se puede tratar de sobrecargar la @.
- No se puede cambiar la prioridad de un operador. Por ejemplo en $a+b*c$, no se puede hacer que primero se sume y luego se multiplique.
- La asociatividad de los operadores no se puede cambiar, esto es !a no se puede usar como a!, o a-> no puede usarse como ->a.

- Los operadores unarios como ++ solo tienen un operando y los otros dos, esto no puede cambiar. Por ejemplo no se puede pretender hacer **a++b** o **a+**.
- Tener en cuenta que no debe haber ambigüedad.

Si se tiene: int a[5];

No se puede tratar de sobrecargar el operador + para hacer a + 3. Porque “a” es un int *, por lo que esa operación ya está contemplada en el Lenguaje C++.

Por lo tanto solo debe sobrecargar los operadores en donde por lo menos un operando sea una estructura o clase.

- Para el caso de las estructuras (**struct**) no se puede sobrecargar el operador **=**.

Plantillas de funciones

DEFINICIÓN:

Es una herramienta del lenguaje C++ que permite implementar una función y que a la hora de compilarla el sistema genere varias versiones de esta función, sobrecargándolas automáticamente de modo que se adapten a los diferentes tipos de datos que se coloquen como parámetros en la función.

Para realizar esta labor, la plantilla requiere implementar un tipo de dato “genérico”. Este tipo de dato será el que cambiará en cada implementación. **template <typename TIPO>**

El compilador revisará el módulo donde se use la plantilla y si por ejemplo en algún punto la función se invoca con enteros y en otra con valores reales, generará dos implementaciones.

Si el archivo que se está compilando no encuentra un llamado a la función (plantilla), el compilador no traducirá la función.

RESTRICCIONES:

El código de una plantilla no puede estar condicionado a un tipo de dato en especial. Esto es, no puede haber una pregunta que pretenda hacer algo diferente si el dato es de un tipo especial.

El usuario de la plantilla debe adaptase a la plantilla, la plantilla no puede adaptarse al tipo de dato que quiere emplear el usuario.

```
T plantilla (T x, T y){  
    T z;  
    if (x > y) z = x - y;  
    else z = y - x;  
    return z;  
}
```

```
char plantilla (char x, char y){  
    char z;  
    if (x > y) z = x - y;  
    else z = y - x;  
    return z;  
}
```

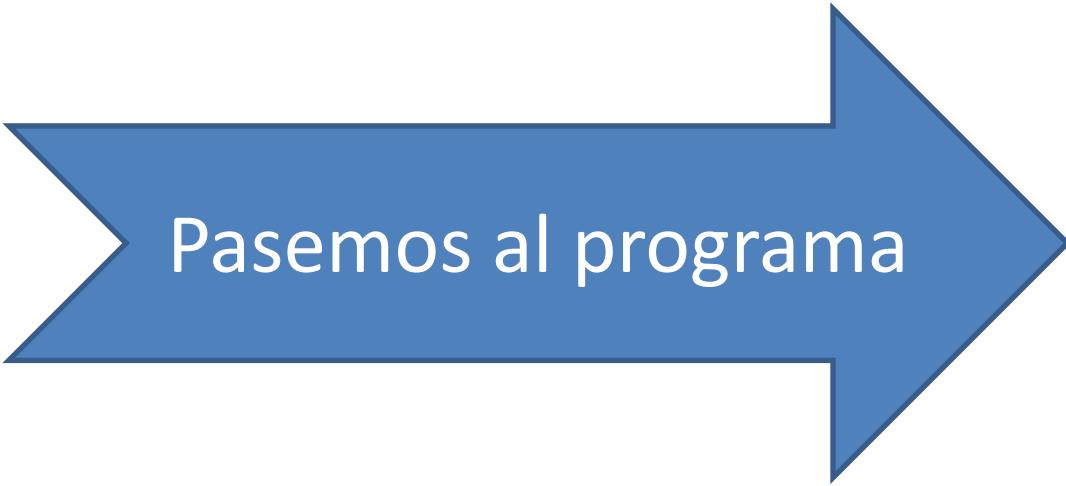
Proyecto

```
#include <plantilla.h>  
int main(){  
    int a, b , c;  
    c = plantilla(a,b);  
    double p,q,r;  
    x = plantilla(q,r);  
    char m,n,k;  
    k= plantilla(m,n);  
    char e[5], f[5] , g[5];  
    e = plantilla(f,g);  
}
```

```
int plantilla (int x, int y){  
    int z;  
    if (x > y) z = x - y;  
    else z = y - x;  
    return z;  
}
```

```
double plantilla (double x,  
                  double y){  
    double z;  
    if (x > y) z = x - y;  
    else z = y - x;  
    return z;  
}
```

```
char* plantilla (char* x, char* y){  
    char* z;  
    if (x > y) z = x - y;  
    ...
```



Pasemos al programa

Bibliotecas estáticas de funciones

DEFINICIÓN:

Una biblioteca es una agrupación de datos, tipos de datos y funcionalidades organizados de forma que sean reutilizables en más de un proyecto.

Existen dos tipos de bibliotecas:

1. Estáticas, archivos con extensiones “.a” o “.lib”, aunque también se pueden considerar los archivos con extensión “.o”.
2. Dinámicas, con extensión “.dll” (dynamic linked libraries).

Las bibliotecas estáticas son ensambladas y enlazadas junto con el programa que las usa, y por tanto forman parte del archivo ejecutable que se crea.

Las bibliotecas dinámicas, se ensamblan por separado y el enlace ocurre durante la ejecución del programa que utiliza la biblioteca.

No todos los lenguajes de programación permiten la creación y el uso de ambos tipos de bibliotecas. El lenguaje C++ permite crear y usar ambos tipos, mientras que Java y C# solo trabajan con bibliotecas dinámicas.

Los archivos con extensión “.a” o “.lib” son parecidos a los archivos con extensión “.o”, con una diferencia que cuando se enlazan un archivo “.o” en un proyecto, se añaden a este último todas las funciones que se escribieron, esto se debe en parte a que en el archivo “.o” sólo se guarda el código compilado de todas las funciones.

Los archivos “.a” guardan, además del código, el nombre de cada función, esto hace que en el momento de enlazarlas, sólo se añaden al programa ejecutable aquellas funciones que realmente se utilizan en el programa, descartándose las demás.

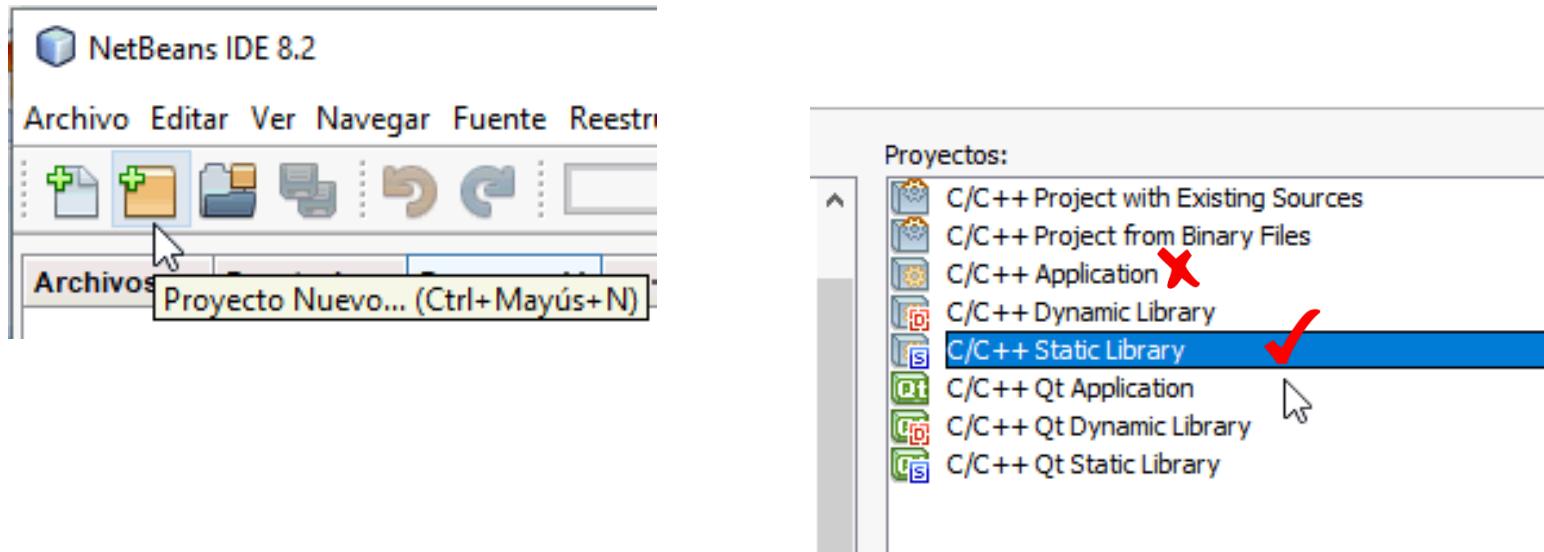
PROCEDIMIENTO A SEGUIR PARA CREAR UNA BIBLIOTECA ESTÁTICA

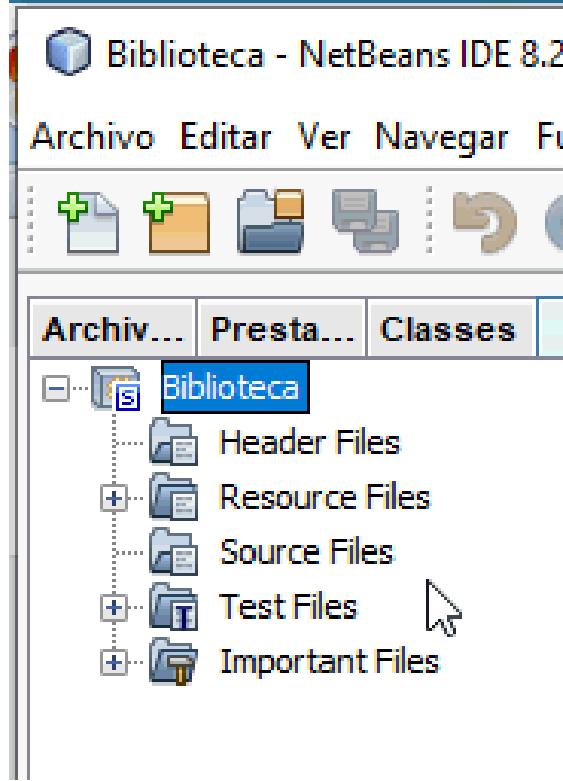
LO PRIMERO que debe saber es que no puede pretender crear la biblioteca (el archivo “.a”) sin antes no tener claro: qué funciones va a colocar en la biblioteca, qué código va a colocar en cada una de ellas y sin antes haber probado ese código.

Por esta razón antes de pensar en crear una biblioteca estática, debe crear un proyecto simple donde realicé todas esas acciones.

Una vez que lo anterior esté hecho:

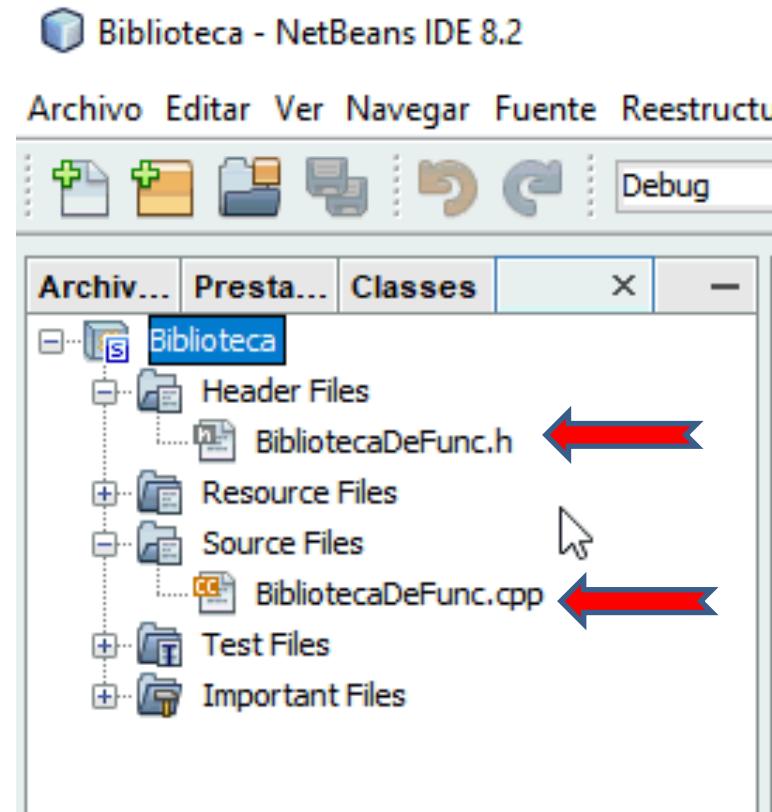
1. Creación del proyecto “BIBLIOTECA ESTÁTICA”



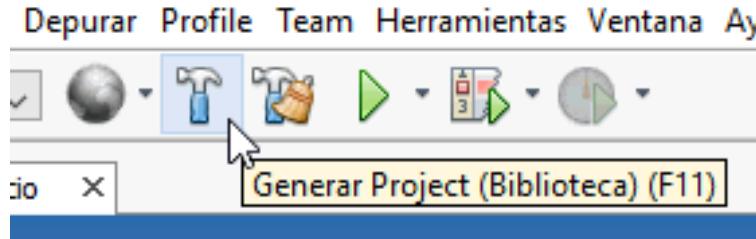


Si el proyecto se creó correctamente, no debe haber haberse creado el archivo main.cpp

Luego debe crear los archivos .h y .cpp para colocar allí sus funciones.



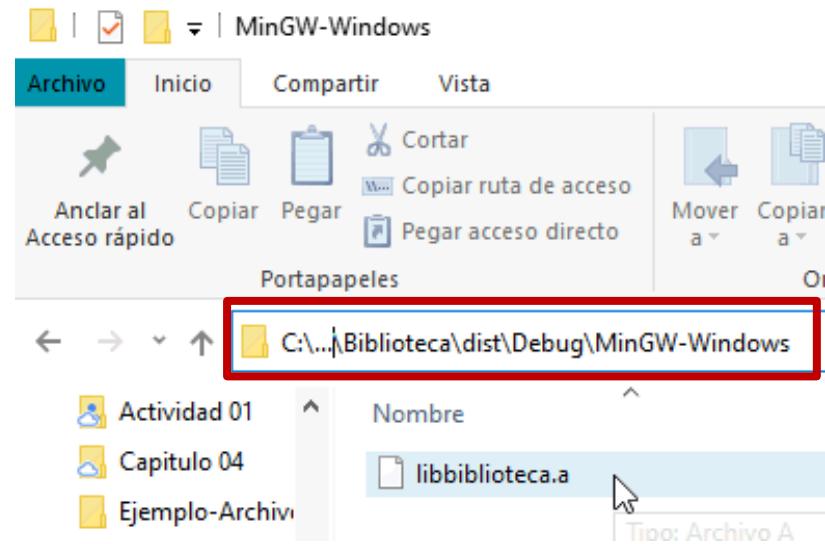
Luego copie los encabezados e implementación de las funciones en el proyecto de prueba y compile este proyecto.



¡RECUERDE QUE A ESTE NIVEL NO SE PRUEBAN LAS FUNCIONES!

2. Creación de un proyecto que use la “BIBLIOTECA ESTÁTICA”

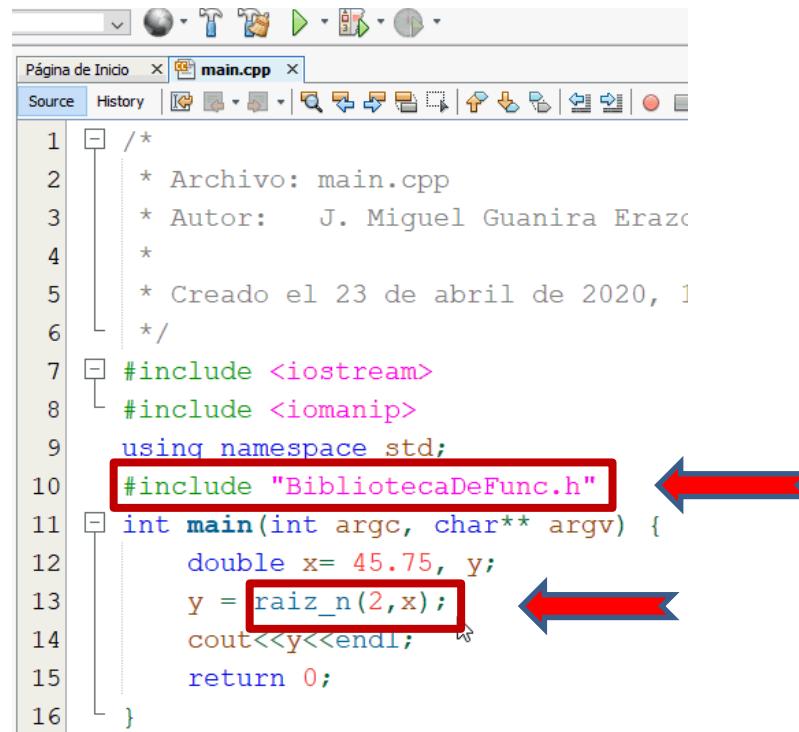
Cree un nuevo proyecto simple (Application) y en él copie el archivo .h y .a de la biblioteca estática.



Escriba el código de su proyecto.

Haga llamados a las funciones de la biblioteca

Y no olvide incluir el archivo .h de la biblioteca

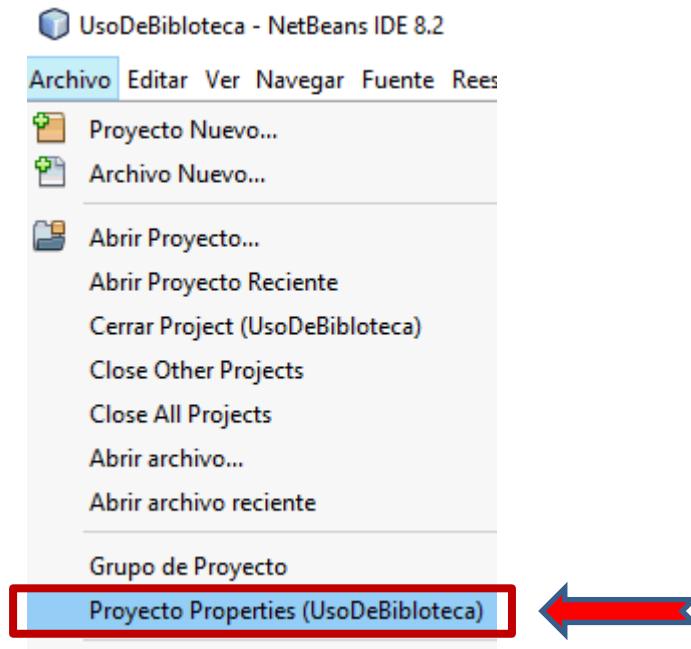


The screenshot shows a code editor window with the file 'main.cpp' open. The code is a C++ program that includes a header file 'BibliotecaDeFunc.h' and calls a function 'raiz_n' from it. Two red arrows point to the inclusion of the header file and the function call.

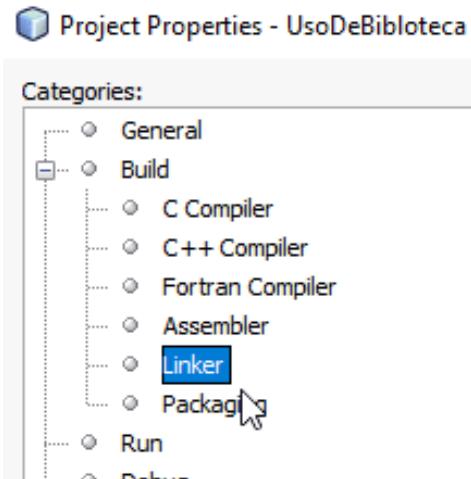
```
1  /*
2   * Archivo: main.cpp
3   * Autor: J. Miguel Guanira Erazo
4   *
5   * Creado el 23 de abril de 2020, 1
6   */
7  #include <iostream>
8  #include <iomanip>
9  using namespace std;
10 #include "BibliotecaDeFunc.h" // Red box and arrow here
11 int main(int argc, char** argv) {
12     double x= 45.75, y;
13     y = raiz_n(2,x); // Red box and arrow here
14     cout<<y<<endl;
15
16 }
```

Incorpore la biblioteca estática en el proyecto.

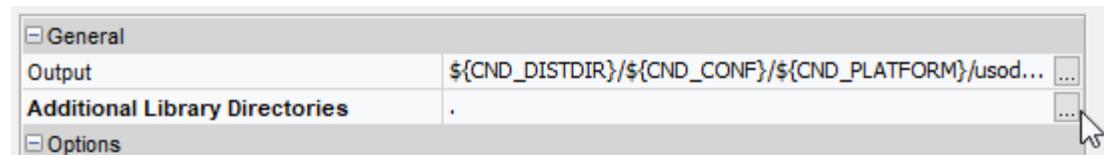
Vaya a las propiedades del proyecto



Elija la opción “Linker”

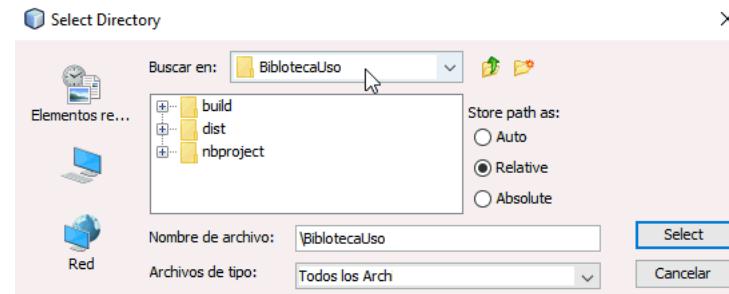


En la opción “Additional Library Directory, presione el botón 

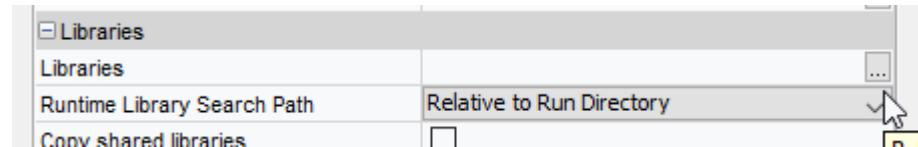


En la ventana que aparece elegir 

Luego seleccione la carpeta donde se encuentra el archivo “.a”

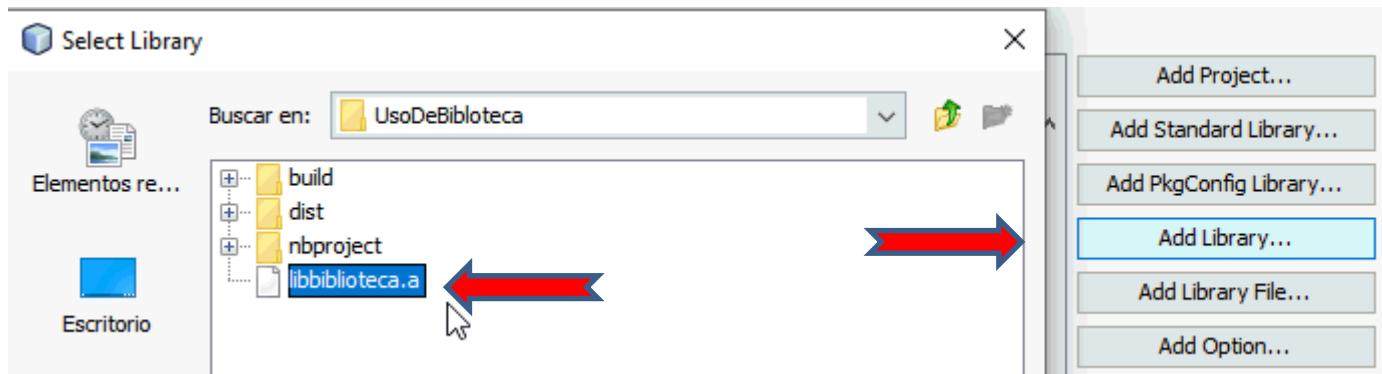


Luego en la opción “Libraries” presione el botón 

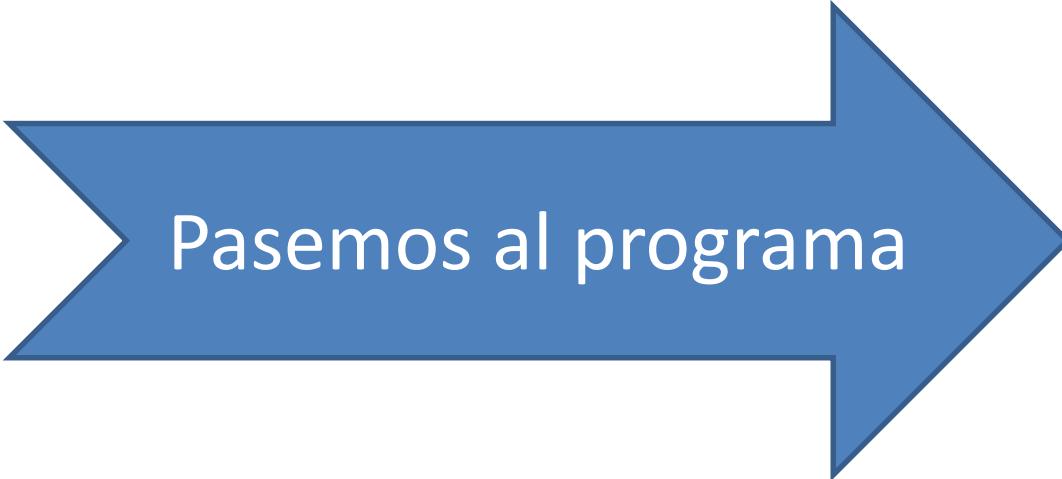


En la ventana que aparece elegir





Allí elegir el archivo “.a”



Pasemos al programa