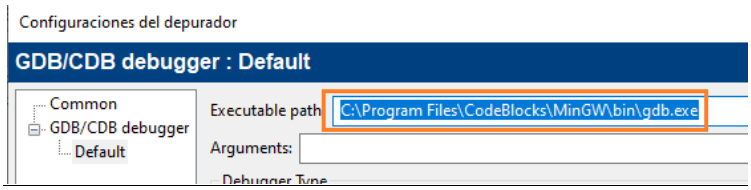
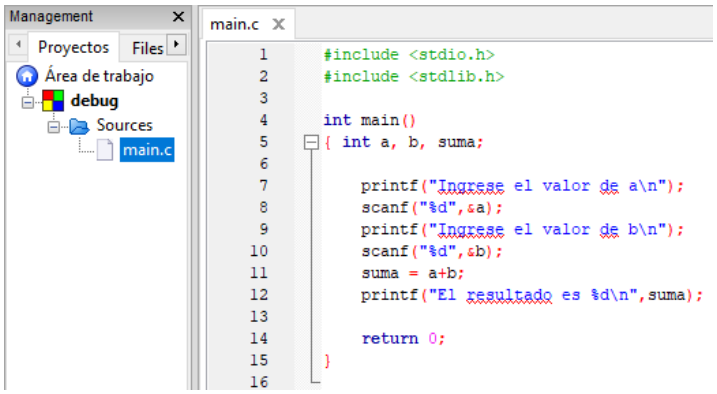
Laboratorio de computación II - Uso del debug en Codeblocks

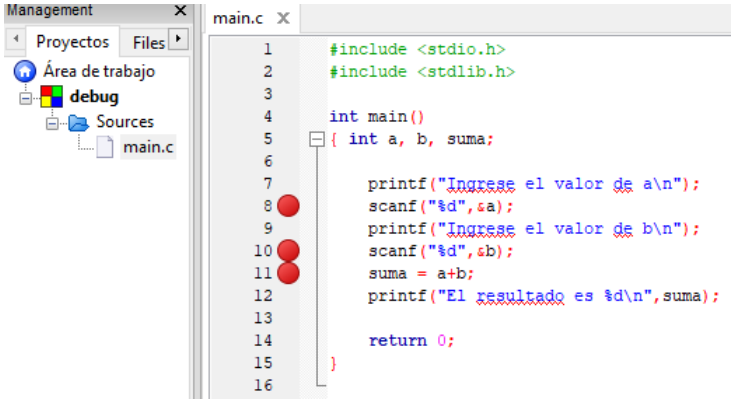
1. Crear un proyecto, **tener en cuenta que en la ruta no debe haber espacios.** Archivo → New → Project, seguir los pasos del asistente. Luego verificar la ruta de gdb.exe, en Preferencias → Debugger→ Default:



2. Construir un programa, por ejemplo:



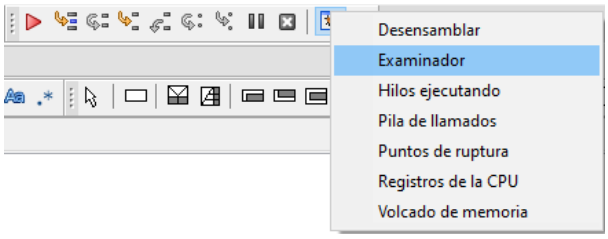
3. Compilarlo. Luego agregar puntos de ruptura, haciendo clic en la línea a la derecha del número del número:



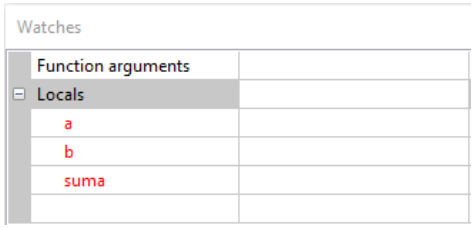
4. En la barra de debug:

5. Pulsar en el botón:

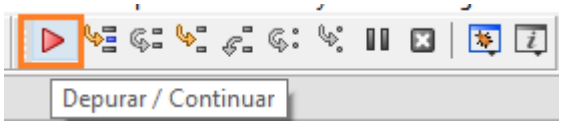
6. Elegir Examinador:



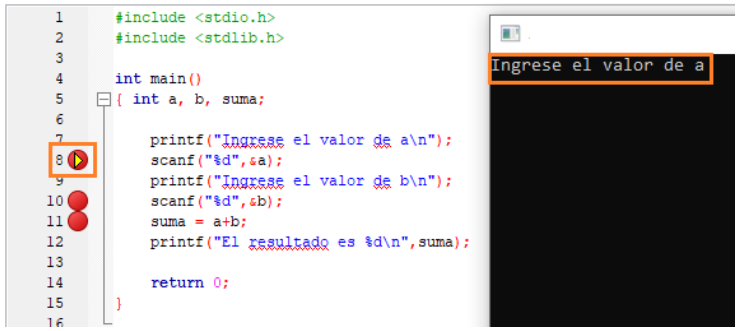
7. Agregar los nombres de las variables:



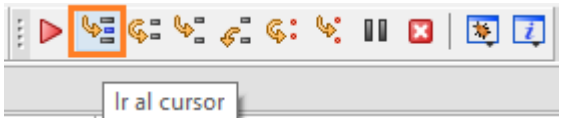
8. Pulsar el botón de depurar:



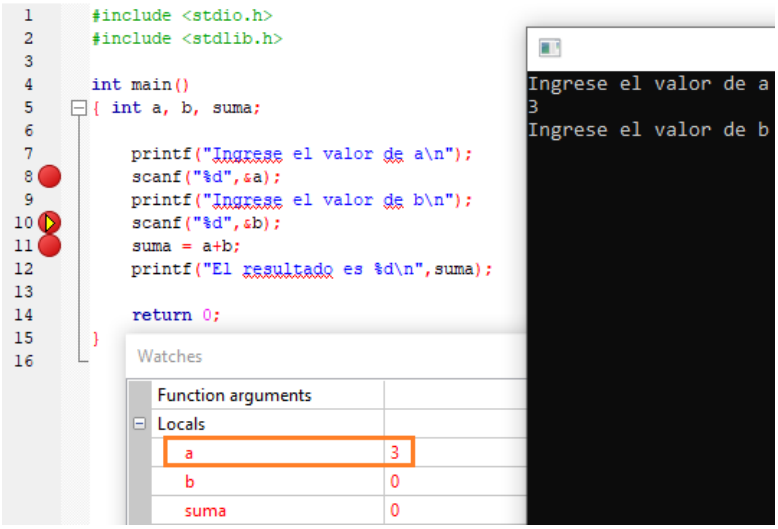
9. El programa se iniciará y se observará una flecha sobre un punto de ruptura:



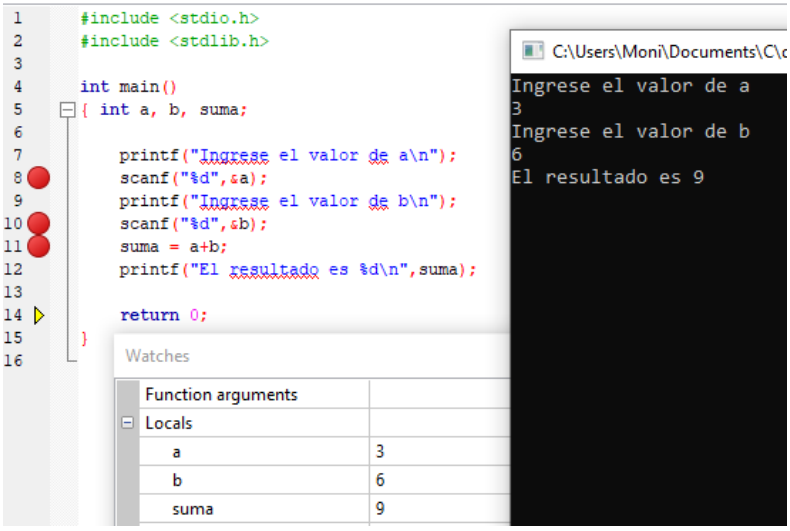
10. El cursor queda esperando pero no se podrá ingresar valores en la ventana de consola hasta que se pulse el siguiente botón o el botón línea siguiente (al lado):



11. Si la ventana Watches desaparece vuelve a activarla y pulsa Enter:



12. Repite los pasos para darle valor a la variable b. Y sigue pulsando el botón indicado hasta finalizar:

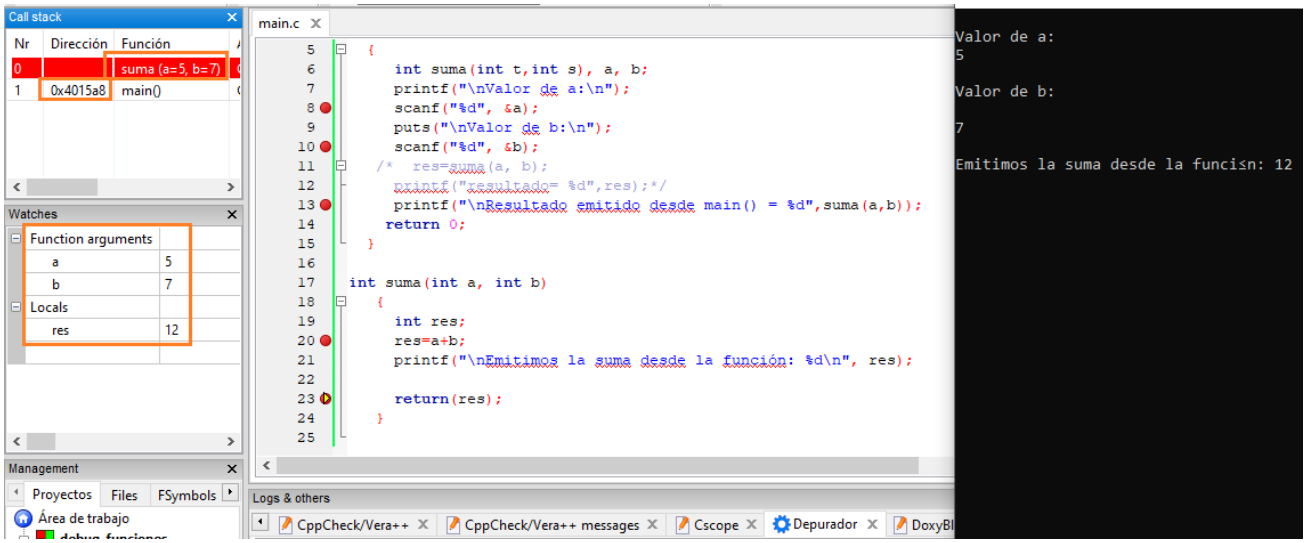


13. En puntos de ruptura pueden verse los marcados en el programa:

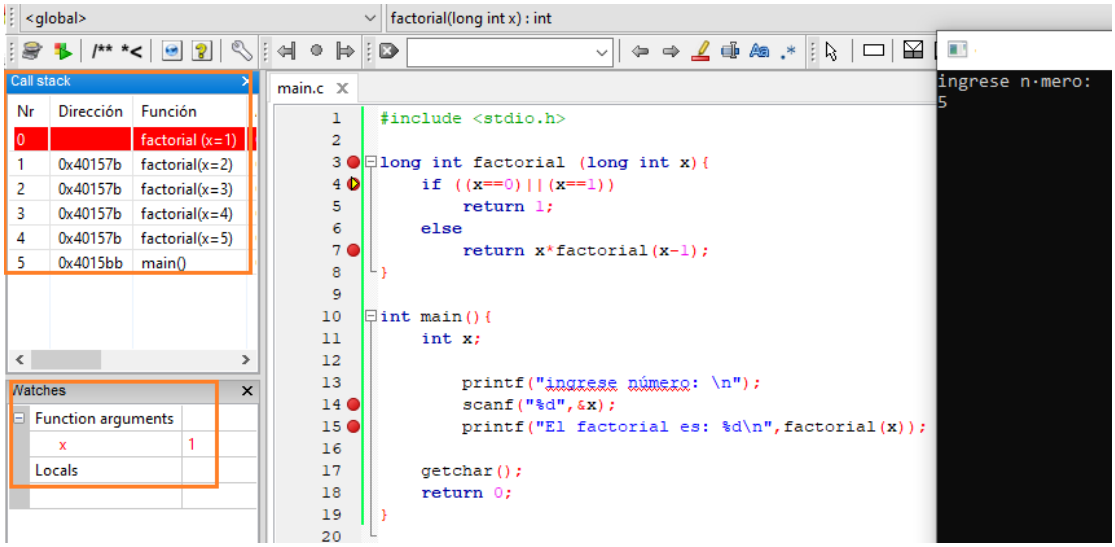
Breakpoints

Tipo	Filename/Address	L...	Informa...	Depurador
Code	\Documents\C\debug\main.c	8	(index: 2)	GDB/CDB debugger
Code	\Documents\C\debug\main.c	10	(index: 3)	GDB/CDB debugger
Code	\Documents\C\debug\main.c	11	(index: 4)	GDB/CDB debugger

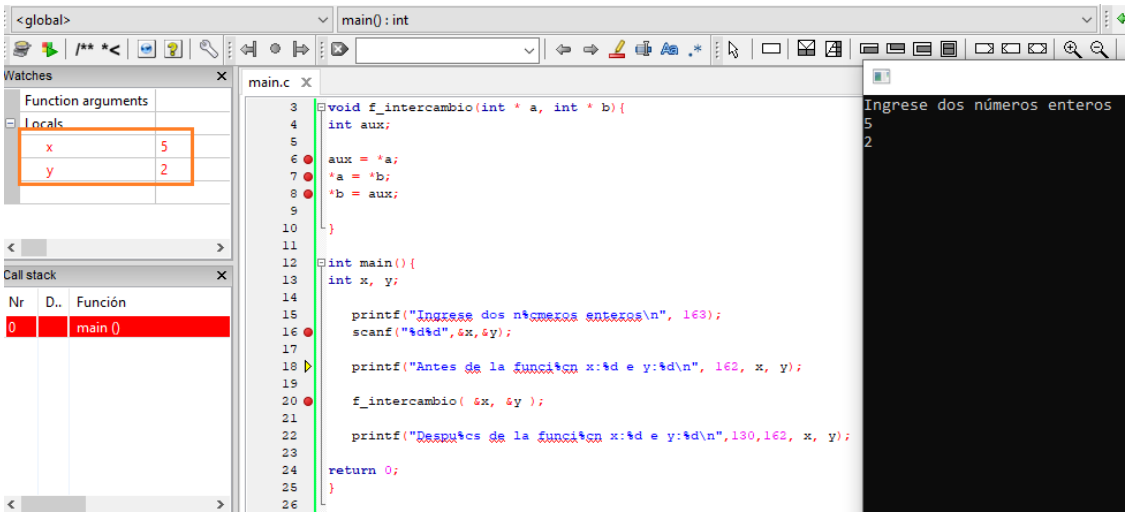
14. Podemos agregar Pila de llamados para revisar las funciones y pasajes de parámetros:

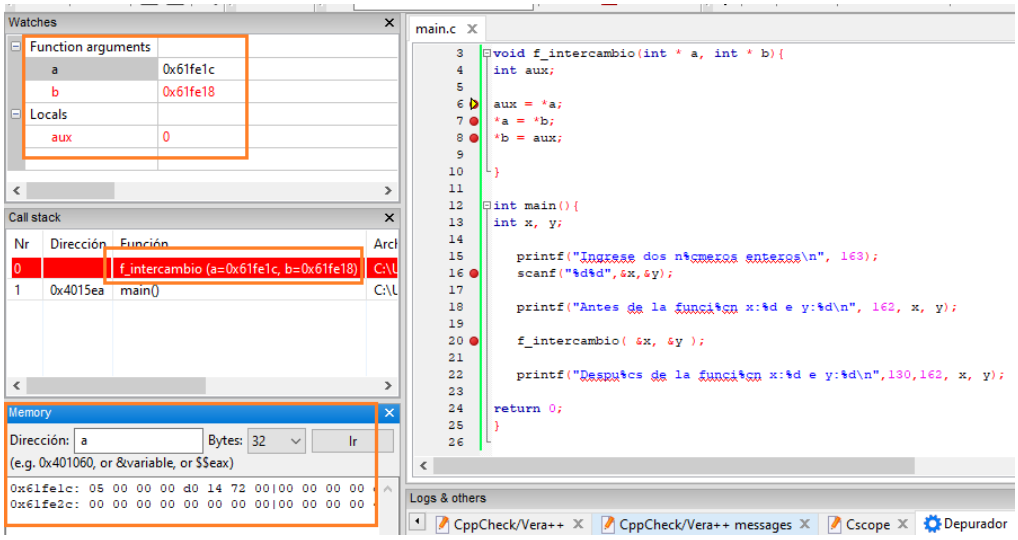


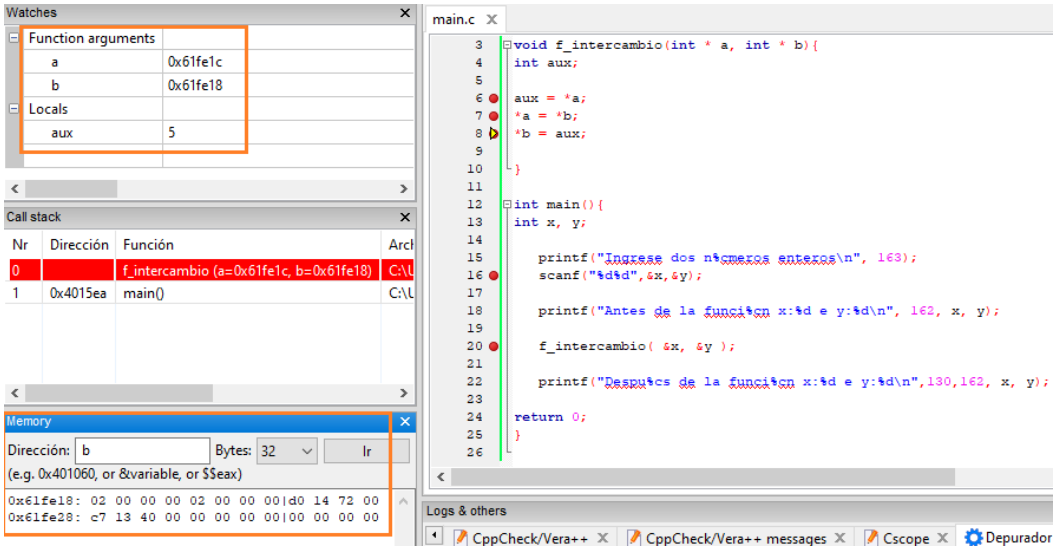
15. En el caso de funciones recursivas podemos los valores que va tomando la variable en cada llamado y ver el apilamiento y desapilamiento en la pila:

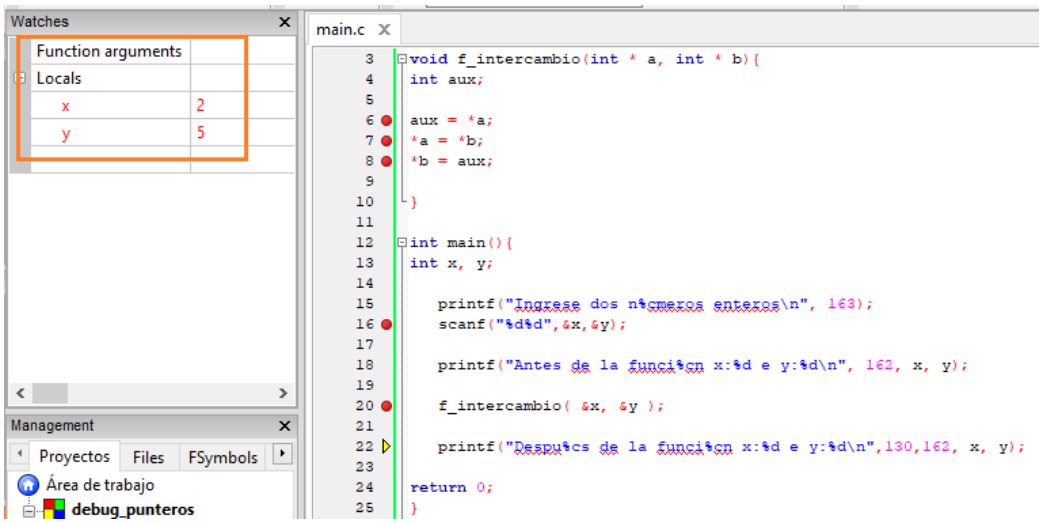


16. Funciones con argumentos por dirección:




Watches
Function arguments
a 0x61fe1c
b 0x61fe18
Locals
aux 0
Call stack
Nr Dirección Función
0 f_intercambio (a=0x61fe1c, b=0x61fe18)
1 0x4015ea main()
Memory
Dirección: a Bytes: 32 Ir
(e.g. 0x401060, or &variable, or $$eax)
0x61fe1c: 05 00 00 00 d0 14 72 00|00 00 00 00
0x61fe2c: 00 00 00 00 00 00 00 00|00 00 00 00
main.c
3 void f_intercambio(int * a, int * b){
4 int aux;
6 aux = *a;
7 *a = *b;
8 *b = aux;
10 }
12 int main(){
13 int x, y;
15 printf("Ingrese dos n%cmeros enteros\n", 163);
16 scanf("%d%d",&x,&y);
18 printf("Antes de la funci%cn x:%d e y:%d\n", 162, x, y);
20 f_intercambio(&x, &y);
22 printf("Despu%cs de la funci%cn x:%d e y:%d\n",130,162, x, y);
24 return 0;
25 }
Logs & others
CppCheck/Vera++ CppCheck/Vera++ messages Cscope Depurador


Watches
Function arguments
a 0x61fe1c
b 0x61fe18
Locals
aux 5
Call stack
Nr Dirección Función
0 f_intercambio (a=0x61fe1c, b=0x61fe18)
1 0x4015ea main()
Memory
Dirección: b Bytes: 32 Ir
(e.g. 0x401060, or &variable, or $$eax)
0x61fe18: 02 00 00 00 02 00 00 00|d0 14 72 00
0x61fe28: c7 13 40 00 00 00 00 00|00 00 00 00
main.c
3 void f_intercambio(int * a, int * b){
4 int aux;
6 aux = *a;
7 *a = *b;
8 *b = aux;
10 }
12 int main(){
13 int x, y;
15 printf("Ingrese dos n%cmeros enteros\n", 163);
16 scanf("%d%d",&x,&y);
18 printf("Antes de la funci%cn x:%d e y:%d\n", 162, x, y);
20 f_intercambio(&x, &y);
22 printf("Despu%cs de la funci%cn x:%d e y:%d\n",130,162, x, y);
24 return 0;
25 }
Logs & others
CppCheck/Vera++ CppCheck/Vera++ messages Cscope Depurador


Watches
Function arguments
Locals
x 2
y 5
Management
Proyectos Files FSymbols
Área de trabajo
debug_punteros
main.c
3 void f_intercambio(int * a, int * b){
4 int aux;
6 aux = *a;
7 *a = *b;
8 *b = aux;
10 }
12 int main(){
13 int x, y;
15 printf("Ingrese dos n%cmeros enteros\n", 163);
16 scanf("%d%d",&x,&y);
18 printf("Antes de la funci%cn x:%d e y:%d\n", 162, x, y);
20 f_intercambio(&x, &y);
22 printf("Despu%cs de la funci%cn x:%d e y:%d\n",130,162, x, y);
24 return 0;
25 }