Ce cours/TD/TP est à réaliser à l'aide de code::block

Disponible gratuitement ici : <u>https://codeblocks.org/downloads/26</u> Souc Windows, télecharger codeblocks-xx.yymingw-setup.exe (xx.yy est le numéro de la dernière version)

code ::block est préinstallé sur les distributions Linux type Debian (Ubuntu, Mint ...)

Cette version de code::block dispose du compilateur c/c++ mingw sur windows et gcc sous Linux ainsi que du debugger GDB (Gnu Debugger) qui permet de poser des points d'arrêt, d'avancer un programme en pas à pas et de consulter les variables.

Prérequis :

Connaissances élémentaires du langage C. main, fonctions, variables, types, printf. Cration d'un projet dans code::block en langage C.

1 Prise en main du debugger

Ecrire le programme p1.c ci dessous, dans code::block qui va permettre de visualiser les divers zones d'enregistrements des variables en mémoire.

Les variables locales (ici xloc et yloc), sont déclarées dans une fonction elles ne sont accessibles que depuis la fonction ou elles sont déclarées (ici main). Les variables globales en dehors des fonctions (ici iglo et jglo) sont accessibles de n'importe quelle partie du programme.

```
char mess[]="Bonjour"; // une chaine
int iglo; // variables globales
int jglo;
int main()
{
    char xloc,yloc; // variables locales
    iglo=0x12345678; // les entiers sont codés sur 32bits
    jglo=0xABCDEF09;
    xloc=0x11; // les char sont codés sur 8 bits
    yloc=0x22;
    return 0;
}
```

Assurez-vous que la syntaxe est bonne, compiler-executer ... il n'y a rien à voir puisque le programme ne fait rien. Le debugger va permettre de visualiser les variables dans la mémoire de l'ordinateur.

Commandes du debbuger



Lancer debug / executer / executer une ligne / entrer dans la fonction / pause / arrêt debug

Placer un point d'arrêt sur la ligne iglo=0x12345678 ; pour cela cliquer à droite du numéro de ligne, un point rouge apparaît. Lancer le debugger puis en cliquant sur le triangle rouge … l'exécution s'arrête sur le point d'arrêt (triangle jaune).

```
1
       #include <stdio.h>
2
       #include <stdlib.h>
3
 4
       char mess[]="Bonjour"; // une chaine
 5
       int iglo;
                    // variables globales
       int jglo;
 6
 7
8
       int main()
9
     - {
10
                               // variables locales
           char xloc, yloc;
11 🗘
           iglo=0x12345678; // les entiers sont codés sur 32bits
           jglo=0xABCDEF09;
12
13
           xloc=0x11;
                         // <u>les</u> char <u>sont</u> codés <u>sur</u> 8 bits
14
           yloc=0x22;
15
           return 0;
16
       - 1
```

Cliquer sur Next Line (executer une ligne) et dérouler le programme en pas à pas jusqu'à la fin. Relancer à nouveau le debugger jusqu'au point d'arrêt.

Pour observer les variables cliquer sur le cafard \rightarrow

8

Faire apparaître les fenêtres « Memory dump » et « Watches»

Waches affiche les valeurs actuelles des variables, remarquer la distinction entre les locales et les globales

W	/atches			x
	Function arguments			
Ξ	Locals			
	xloc	0 '\000'		
	yloc	0 '\000'		
	iglo	0	int	
	jglo	0	int	
	mess	"Bonjour"	char [8]	

Dérouler le programme en pas à pas jusqu'à return 0 ;,

La fenêtre **« Watches »** affiche les valeurs des variables à la fin du programme. *Remarque : la fenêtre « Watches » affiche les variables en décimal, un clic-droit – Properties permet un affichage hexadecimal.*

Remarque : Il est possible de modifier la valeur d'une variable dans Watches par un double clic sur sa valeur. L'accès est possible hors de la zone "locals »

W	atches		x										
	Function arguments												
⊡	Locals												
	xloc	17 '\021'											
	yloc	34 ""											
	iglo	0x12345678	int										
	jglo	0xabcdef09	int										
	xloc	0x11	char										
	yloc	0x22	char										
	mess	"Bonjour"	char [8]										

Memory permet de visualiser (dumper) le contenu de la mémoire. Pour visualiser la variable iglo, entrer dans la barre « Address » **&iglo.**

La fenêtre « Memory » ressemble alors à celle-ci

Memory est divisé en trois parties, à gauche les adresses, au centre les valeurs en hexadécimal, à droite les valeurs en ASCII.

Memory																	×
Address: 8jg	jlo 50 o	vr 804	ariał		vr éé	(veo				/						Bytes: 256 🗸 Go	
(e.g. 0x4010	00,0		ana	ле, (n da	сах											
0x405050:	09	ef	cd	ab	78	56	34	12 00	00	00	00	00	00	00	00	.ïÍ«xV4	\sim
0x405060:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x405070:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x405080:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x405090:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4050a0:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4050b0:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4050c0:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4050d0:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4050e0:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		\checkmark
<																2	

Les adresses sont codées sur 32 bits donc quatre octets. L'octet de poids fort étant ici égale à 0, il n'est pas représenté, (0x405050 est en fait 0x00405050).

Les données sont affichées par groupes de 16 octets ainsi la première ligne affiche les contenus des adresses 0x405050 à 0x40405F

Les variables sont rangées l'une après l'autre mais « à l'envers » . Ce type de stockage s'appelle « little endian »



iglo est rangée à l'adresse 0x405050 (quatre octets) (iglo = 0xabcdef09) jglo est rangée à l'adresse 0x405054 (quatre octets) (jglo = 0x12345678)

Dans la barre d'adresse de « Memory » entrer l'adresse de yloc, relever alors l'adresse et la valeur de xloc et yloc

Memory														_		
Address: &yl	oc								_	_	_					Bytes: 256 🗸
(e.g. 0x401060, or &variable, or \$\$eax)																
0x60fefe:	22	11	83	00	00	00	02	00100	00	80	ff	60	00	fd	10	"ÿ`.ý.
0x60ff0e:	40	00	01	00	00	00	d0	16 69	00	78	le	69	00	00	50	@Ð.i.x.iP
0x60ffle:	40	00	40	ff	60	00	ff	ff ff	ff	44	ff	60	00	c0	cc	@.@ÿ`.ÿÿÿÿDÿ`.ÀÌ
0x60ff2e:	14	75	41	52	d5	le	fe	ff ff	ff	44	ff	60	00	67	ca	.uARÕ.þÿÿÿDÿ`.gÊ
0x60ff3e:	14	75	78	le	69	00	00	00100	00	0d	76	14	75	01	00	.ux.iv.u
0x60ff4e:	00	00	00	a 0	35	00	95	12 40	00	01	00	00	00	00	00	5@
0x60ff5e:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00	
0x60ff6e:	00	00	00	00	00	00	59	63 80	76	00	a 0	35	00	40	63	Yev. 5.@c
0x60ff7e:	80	76	dc	ff	60	00	74	7b 6c	77	00	a 0	35	00	08	52	vÜÿ`.t{lw. 5R
0x60ff8e:	61	ь0	00	00	00	00	00	00100	00	00	a0	35	00	00	00	a° 5

Faire de même pour la chaîne de caractère mess, qui ici est rangée aux adresses 0x402000 à 0x402007, soit huit octets, le 0x00 à l'adresse 0x402007 indique la fin de la chaine ASCII.

Memor	у																	x
Address:	8m	ess					\swarrow										Bytes: 256 \checkmark Go	
(e.g. 0x4	ю 106	50, o	r &v	ariab	ole, d	or \$\$	ieax]											
0x4020	00:	42	6f	6e	6a	6f	75	72	00 ff	ff	ff	ff	00	40	00	00	Bonjour.ÿÿÿÿ.@	~
0x4020	10:	60	lc	40	00	00	00	00	00100	00	00	00	00	00	00	00	`.@	
0x4020	20:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4020	30:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4020	40:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4020	50:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4020	60:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4020	70:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4020	80:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		
0x4020	90:	00	00	00	00	00	00	00	00100	00	00	00	00	00	00	00		~
<																		> .;

Les variables de type pointeurs en C/C++

Les pointeurs sont des variables qui désigne l'adresse d'une variable.

Les pointeurs ont toujours la même taille (la taille d'une adresse) mais peuvent pointer sur des variables de types différents.

On peut créer des pointeurs sur des char, int, float, tableau etc....

Exemple :

soit p un pointeur sur des entiers courts (short 16 bits), pour le déclarer on ajoute une * devant le nom Soit i un entier court (short 16bits)

Le programme ci-dessous permet avec le debugger de visualiser j et	t p
--	-----

1		<pre>#include <stdio.h></stdio.h></pre>	_									\leq	/								
2			Memory																	x	
3		short int i;							/		\square									_	
4		<pre>short int *p;</pre>	Address:	&i			/	\leq		/									Bytes: 256 V Go		
5			(e.g. 0x40	1060) <mark>, o</mark> r	&var	iable	, or \$	\$eax)											
6		int main(void)	0x40505	0: 3	34 1	2 0	0 0	0 5	50	40	00100	00	00	00	00	00	00	00	4PP@	~	5
7	Ę	l C	0x40506	0:	00 0	0 0	0 0	0 0	0 00	00	00100	00	00	00	00	00	00	00			1
8		i=0x1234;	0x40507	0:	00 0	0 0	0 0	0 0	0 00	00	00100	00	00	00	00	00	00	00			
9		p=&i	0x40508	0:	00 0	0 0	0 0	0 0	00 00	00	00100	00	00	00	00	00	00	00			
10		<pre>printf("%p\n",p);</pre>	0x40509	0:				0 0	000	00	00100	00	00	00	00	00	00	00			
11 🚺		return 0;	0x4050b	0:	00 0	0 0	0 0	0 0	00 0	00	00100	00	00	00	00	00	00	00		- 17	1
12		}	0x4050c	0:	00 0	0 0	0 0	0 0	0 00	00	00100	00	00	00	00	00	00	00			
13	L		0x4050d	0:	00 0	0 0	0 0	0 0	0 00	00	00100	00	00	00	00	00	00	00			
			0x4050e	0:	00 0	0 0	0 0	0 0	0 00	00	00100	00	00	00	00	00	00	00		~	1
			<																	> .,	4

i se trouve à l'adresse 0x00405050 et contient 0x1234 (codage little endian)

p se trouve à l'adresse 0x00405054. Ici une adresse est composée de quatre octets, donc p occupe quatre octets.

p contient 0x00405040 qui est bien l'adresse de i, on dit que p pointe sur i.

2 Exercice , prise en main

Dérouler le programme suivant en pas à pas, observer l'évolution des variables dans les fenêtres watches et memory.

Remarquer en particulier dans la fenêtre memory la recopie du message

```
#include <stdio.h>
#include <stdlib.h>
char source[]="Bonjour";
char destination[]="0000000000";
int recopie(char *s, char *d)
{
int cpt=0;
 while (s!='(0')
 {
    *d=*s;
    s++;
    d++;
    cpt++;
 }
 *d='\0';
 cpt++;
 return cpt;
}
int main()
{
int nb;
  nb=recopie(source,destination);
  printf("recopie de %d caracteres %s vers %s\n",nb,source,destination);
  return 0;
}
```



s et d sont des pointeurs (des adresses) sur des char.

*s et *d représentent les contenus de ces adresses

Ici s pointe sur le caractère ASCII 0x6A 'j' du tableau source et d pointe sur le caractère ASCII 0x6A 'j' du tableau destination

3 Fonctionnalités avancées du debugger

Le debugger permet d'observer et modifier le programme en profondeur, jusqu'au code machine.

Placer un point d'arrêt sur le while de la fonction recopie. Ajouter les fenêtres de debug Disassembly et CPU Registers.

Executer le programme jusqu'au point d'arrêt, la fenêtre Disasembly affiche le code de la fonction en langage assembleur et la fenêtre CPE Register l'état des registres du processeur.

Le pas à pas permet de visualiser le déroulement du programme et l'évolution des registres. En observant les registres vous pourrez visualiser les passages des données des tableaux dans ces registres.

Exemple : chargement du code ASCII 0x65 soit 'j' dans le registre eax.

Disassembly				CPU Registe	trs		
Function: recopie (D: Frame start: 0x60fee0	MEGA \Onel	Drive\Documents\DOSSIERS PEDA	GO\Informatique\	Register	Hex	Interpreted	
0x401350	push	%ebp	^	eax	0x6a	106	
0x401351	mov	%esp,%ebp		ecx	0x1	1	
0x401353	sub	\$0x10,%esp		ody	0.750	110	
0x401356	movl	\$0x0,-0x4(%ebp)		eax	OXCE	110	
0x40135d	jmp	0x401376 <recopie+38></recopie+38>		ebx	0x206000	2121728	
> 0x40135f	mov	0x8(%ebp),%eax		esp	0x60fec8	0x60fec8	
0x401362	movzbl	(%eax),%edx		ebp	0x60fed8	0x60fed8	
0x401365	mov	Oxc(%ebp),%eax			0+401280	4199040	
0x401368	mov	<pre>%dl,(%eax)</pre>		est	08401200	4155040	
0x40136a	addl	\$0x1,0x8(%ebp)		edi	0x401280	4199040	
0x40136e	addl	\$0x1 , 0xc(%ebp)		eip	0x40135f	0x40135f <recopie+15></recopie+15>	
0x401372	addl	\$0x1 ,-0x4(%ebp)		eflags	0x206	ותדקי	
0x401376	mov	0x8(%ebp),%eax			000		
0x401379	movzbl	(%eax),%eax		CS	0x23	35	
0x40137c	test	<pre>%al,%al</pre>		55	0x2b	43	
0x40137e	jne	0x40135f <recopie+15></recopie+15>		ds	0x2b	43	
0x401380	mov	Oxc(%ebp),%eax			0x2b	43	
0x401383	movb	\$0x0,(%eax)		-	0425		
0x401386	addl	\$0x1 ,-0x4(%ebp)		fs	0x53	83	
0x40138a	mov	-0x4(%ebp),%eax		gs	0x2b	43	
0x40138d	leave		~				
<			>				
Mixed Mode	Adjust	Save to text file					

4 Gestion de la pile d'appel des fonctions.

Le programme ci-dessous possède une fonction qui calcule le factoriel d'un nombre. (<u>https://fr.wikipedia.org/wiki/Factorielle</u>)

La fonction factorielle est récursive, elle s'appelle elle-même. Ce type de programmation est très efficace du point de vue de la taille du code mais dangereux du point de vue de l'occupation mémoire RAM. La fenêtre « Call stack » du debugger permet de visualiser les appels récursifs.

```
#include <stdio.h>
#include <stdlib.h>
unsigned long factoriel(unsigned int n)
{
 printf("%d ",n);
 if (n == 1 || n == 0) return 1;
 printf("x ");
 return(n*factoriel(n - 1)); // appel recursif ici
}
                                                      Call stack
int main()
                                                      Nr
                                                           Address
                                                                    Function
                                                                                 File
                                                                                                Line
{
                                                                   factoriel (n=1) D:\M...
                                                      0
unsigned int nb;
                                                           0x401377 factoriel(n=2) D:\M...
                                                                                                   7
                                                      1
unsigned long fac;
                                                           0x401377 factoriel(n=3)
                                                                                                   7
                                                      2
                                                                                 D:\M...
  nb=4;
                                                                                                  7
                                                      3
                                                           0x401377 factoriel(n=4)
                                                                                 D:\M...
  fac=factoriel(nb);
                                                      4
                                                           0x4013a3 main()
                                                                                 D:\M...
                                                                                                  15
  printf("-> %d!= %ld\n",nb,fac);
  return 0:
}
                               ID:\MEGA\OneDrive\Documents\DOSSIERS PEDAGO
                                      3
                                          x 2 x 1 ->
                                                                 4! = 24
```

Process returned 0 (0x0) Press any key to continue.

Placer un point d'arrêt sur nb=4 ; et dérouler le programme en pas à pas afin d'observer l'évolution des appels récursifs.