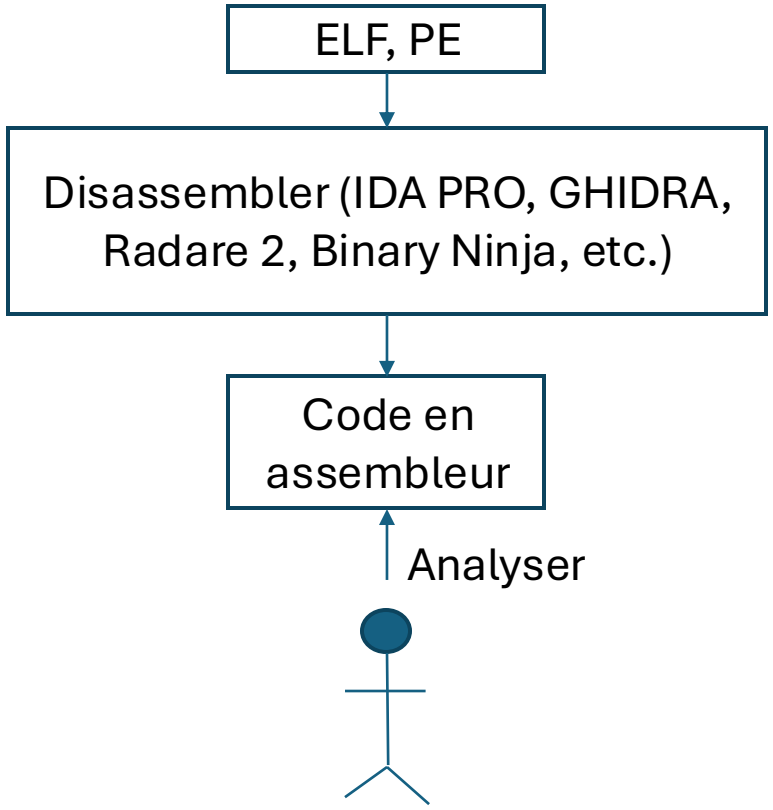
# Reconnaissance des constructions C dans le code en assembleur avec IDA PRO

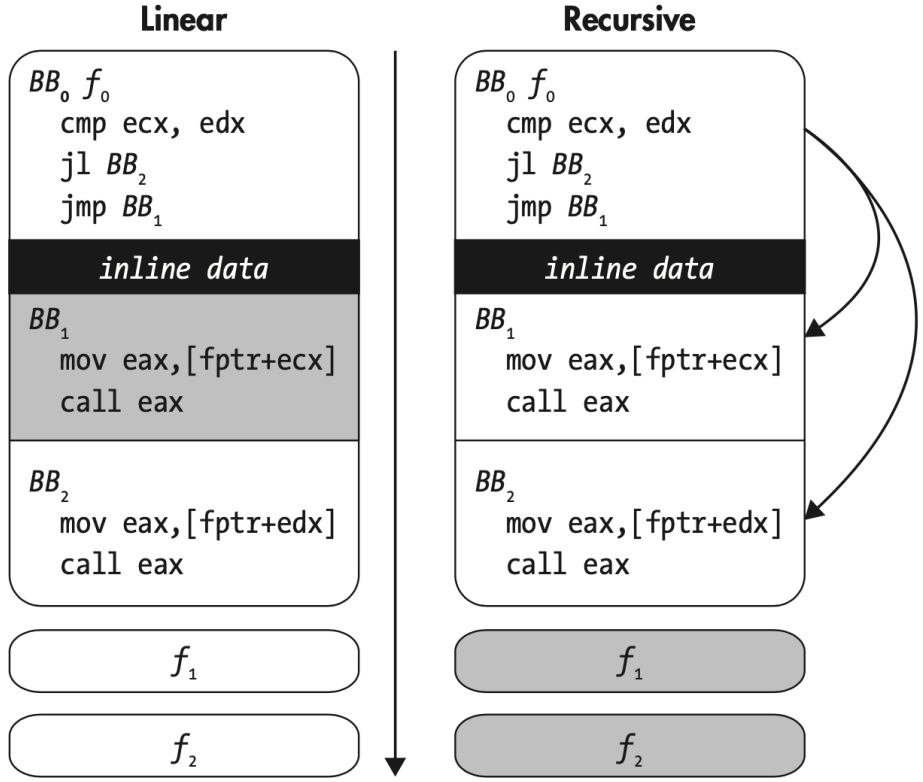
# Désassembleur

- Retrouver le code assembleur à partir d'un fichier binaire (ELF, PE)
- **Linear Sweep Disassembly**
  - Couvre la totalité du code
  - Ne gère pas du code inclus dans le code
- **Recursive disassembly**
  - Permet de contourner les données incluses dans le code
  - C'est la méthode par défaut de la plupart des désassembleur comme IDA-Pro
  - Plus compliquée à réaliser (ex. gérer les sauts indirects)



# Désassembleur

- Retrouver le code assembleur à partir d'un fichier binaire (ELF, PE)
- **Linear Sweep Disassembly**
  - Couvre la totalité du code
  - Ne gère pas du code inclus dans le code
- **Recursive disassembly**
  - Permet de contourner les données incluses dans le code
  - C'est la méthode par défaut de la plupart des désassembleur comme IDA-Pro
  - Plus compliquée à réaliser (ex. gérer les sauts indirects)



# Variables Globales Vs Variables locales

- Les variable globales sont référencées par des adresses mémoires
- Les variables locales sont référencées par rapport à EBP ou ESP

```
00401003        mov     eax, dword_40CF60
00401008        add     eax, dword_40C000
0040100E        mov     dword_40CF60, eax ❶
00401013        mov     ecx, dword_40CF60
00401019        push    ecx
0040101A        push    offset aTotalD  ;"total = %d\n"
0040101F        call    printf
```

```
00401006        mov     dword ptr [ebp-4], 0
0040100D        mov     dword ptr [ebp-8], 1
00401014        mov     eax, [ebp-4]
00401017        add     eax, [ebp-8]
0040101A        mov     [ebp-4], eax
0040101D        mov     ecx, [ebp-4]
00401020        push    ecx
00401021        push    offset aTotalD  ; "total = %d\n"
00401026        call    printf
```

```
00401006        mov     [ebp+var_4], 0
0040100D        mov     [ebp+var_8], 1
00401014        mov     eax, [ebp+var_4]
00401017        add     eax, [ebp+var_8]
0040101A        mov     [ebp+var_4], eax
0040101D        mov     ecx, [ebp+var_4]
00401020        push    ecx
00401021        push    offset aTotalD  ; "total = %d\n"
00401026        call    printf
```

# Construction if

- Un saut conditionnel doit exister, mais pas tous les sauts conditionnels conduisent à une construction if

```
int x = 1;
int y = 2;

if(x == y){
      printf("x equals y.\n");
}else{
      printf("x is not equal to y.\n");
}
```

```
00401006          mov     [ebp+var_8], 1
0040100D          mov     [ebp+var_4], 2
00401014          mov     eax, [ebp+var_8]
00401017          cmp     eax, [ebp+var_4] ❶
0040101A          jnz     short loc_40102B ❷
0040101C          push    offset aXEqualsY_ ; "x equals y.\n"
00401021          call    printf
00401026          add     esp, 4
00401029          jmp     short loc_401038 ❸
0040102B loc_40102B:
0040102B          push    offset aXIsNotEqualToY ; "x is not equal to y.\n"
00401030          call    printf
```

# Construction if

- IDA PRO en mode graphique permet de mieux visualiser des constructions if complexes

```
int x = 0;
int y = 1;
int z = 2;

if(x == y){
     if(z==0){
          printf("z is zero and x = y.\n");
     }else{
          printf("z is non-zero and x = y.\n");
     }
}else{
     if(z==0){
          printf("z zero and x != y.\n");
     }else{
          printf("z non-zero and x != y.\n");
     }
}
```

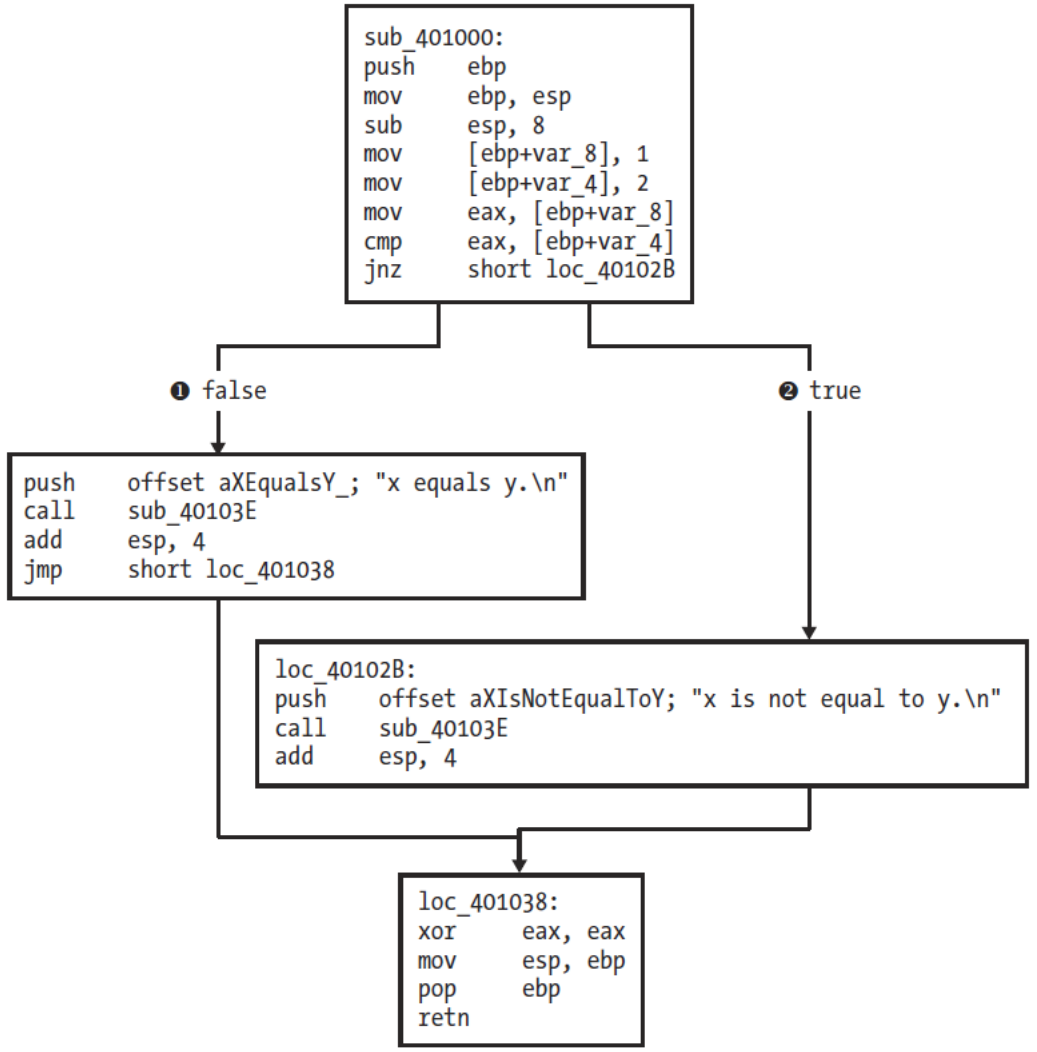
# Construction if

- IDA PRO en mode graphique permet de mieux visualiser des constructions if complexes

```
00401006        mov     [ebp+var_8], 0
0040100D        mov     [ebp+var_4], 1
00401014        mov     [ebp+var_C], 2
0040101B        mov     eax, [ebp+var_8]
0040101E        cmp     eax, [ebp+var_4]
00401021        jnz     short loc_401047 ❶
00401023        cmp     [ebp+var_C], 0
00401027        jnz     short loc_401038 ❷
00401029        push    offset aZIsZeroAndXY_ ; "z is zero and x = y.\n"
0040102E        call    printf
00401033        add     esp, 4
00401036        jmp     short loc_401045
00401038 loc_401038:
00401038        push    offset aZIsNonZeroAndX ; "z is non-zero and x = y.\n"
0040103D        call    printf
00401042        add     esp, 4
00401045 loc_401045:
00401045          jmp     short loc_401069
00401047 loc_401047:
00401047        cmp     [ebp+var_C], 0
0040104B        jnz     short loc_40105C ❸
0040104D        push    offset aZZeroAndXY_ ; "z zero and x != y.\n"
00401052        call    printf
00401057        add     esp, 4
0040105A        jmp     short loc_401069
0040105C loc_40105C:
0040105C        push    offset aZNonZeroAndXY_ ; "z non-zero and x != y.\n"
00401061        call    printf00401061
```

# Construction if

- IDA PRO en mode graphique permet de mieux visualiser des constructions if complexes
- Le lien true en vert et le le lien faux en rouge



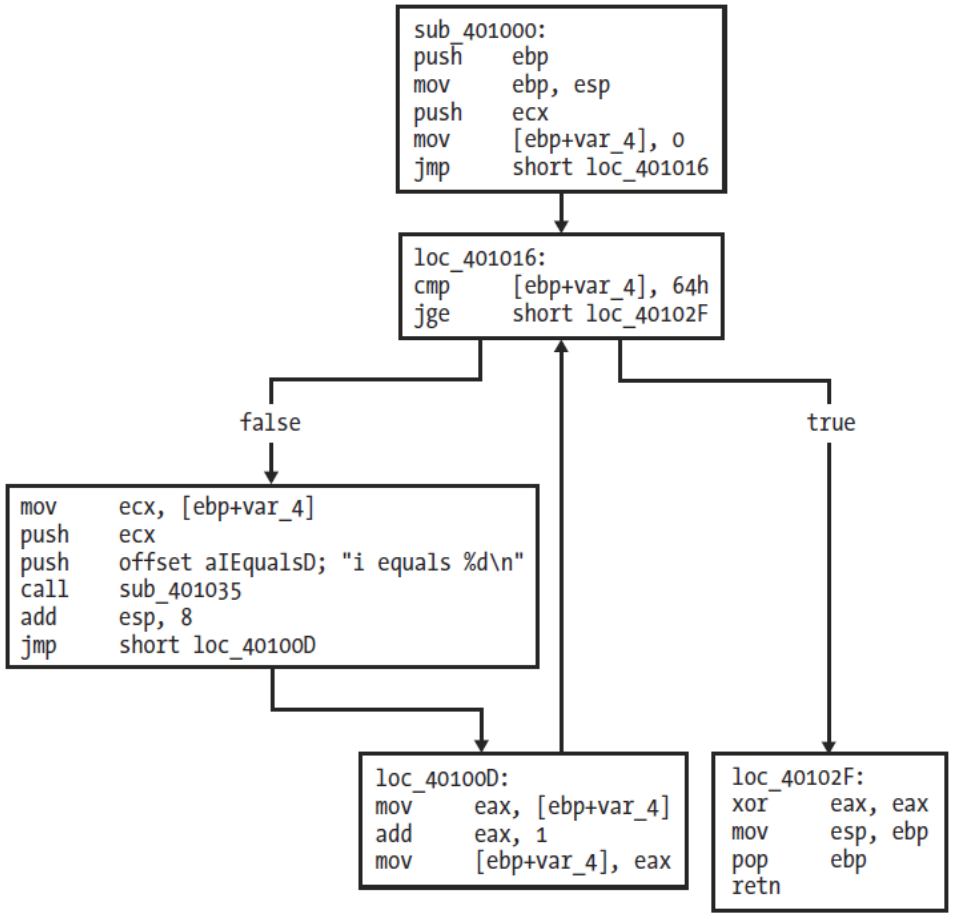
# Boucle for

- La boucle est reconnue en identifiant les étapes:
  - Initialisation (1)
  - Incrémentation (3-4)
  - Comparaison (5) et (6)
  - Le saut inconditionnel (7)

```
00401004          mov     [ebp+var_4], 0 ❶
0040100B          jmp     short loc_401016 ❷
0040100D loc_40100D:
0040100D          mov     eax, [ebp+var_4] ❸
00401010          add     eax, 1
00401013          mov     [ebp+var_4], eax ❹
00401016 loc_401016:
00401016          cmp     [ebp+var_4], 64h ❺
0040101A          jge     short loc_40102F ❻
0040101C          mov     ecx, [ebp+var_4]
0040101F          push    ecx
00401020          push    offset aID  ; "i equals %d\n"
00401025          call    printf
0040102A          add     esp, 8
0040102D          jmp     short loc_40100D ❼
```

# Boucle for

- La boucle est reconnue en identifiant les étapes:
  - Initialisation (1)
  - Incrémentation (3-4)
  - Comparaison (5) et (6)
  - le saut inconditionnel (7)



# Boucle while

- La boucle **while** est très souvent utilisée par les malware pour, par exemple, attendre un événement particulier telle que la réception d'une commande ou d'un paquet
- Même structure que la boucle for sans l'incrémentation

# Construction switch

- Deux manière de compiler le switch
  - Style if
  - Jump table

# Construction switch

- Style if

```
switch(i)
{
   case 1:
      printf("i = %d", i+1);
      break;
   case 2:
      printf("i = %d", i+2);
      break;
   case 3:
      printf("i = %d", i+3);
      break;
   default:
      break;
}
```

```
00401013         cmp     [ebp+var_8], 1
00401017         jz      short loc_401027 ❶
00401019         cmp     [ebp+var_8], 2
0040101D         jz      short loc_40103D
0040101F         cmp     [ebp+var_8], 3
00401023         jz      short loc_401053
00401025         jmp     short loc_401067 ❷
00401027 loc_401027:
00401027         mov     ecx, [ebp+var_4] ❸
0040102A         add     ecx, 1
0040102D         push    ecx
0040102E         push    offset unk_40C000 ; i = %d
00401033         call    printf
00401038         add     esp, 8
0040103B         jmp     short loc_401067
0040103D loc_40103D:
0040103D         mov     edx, [ebp+var_4] ❹
00401040         add     edx, 2
00401043         push    edx
00401044         push    offset unk_40C004 ; i = %d
00401049         call    printf
0040104E         add     esp, 8
00401051         jmp     short loc_401067
00401053 loc_401053:
00401053         mov     eax, [ebp+var_4] ❺
00401056         add     eax, 3
00401059         push    eax
0040105A         push    offset unk_40C008 ; i = %d
0040105F         call    printf
00401064         add     esp, 8
```

# Construction switch

- Jump table

```
switch(i)
{
   case 1:
      printf("i = %d", i+1);
      break;
   case 2:
      printf("i = %d", i+2);
      break;
   case 3:
      printf("i = %d", i+3);
      break;
   case 4:
      printf("i = %d", i+3);
      break;
   default:
      break;
}
```

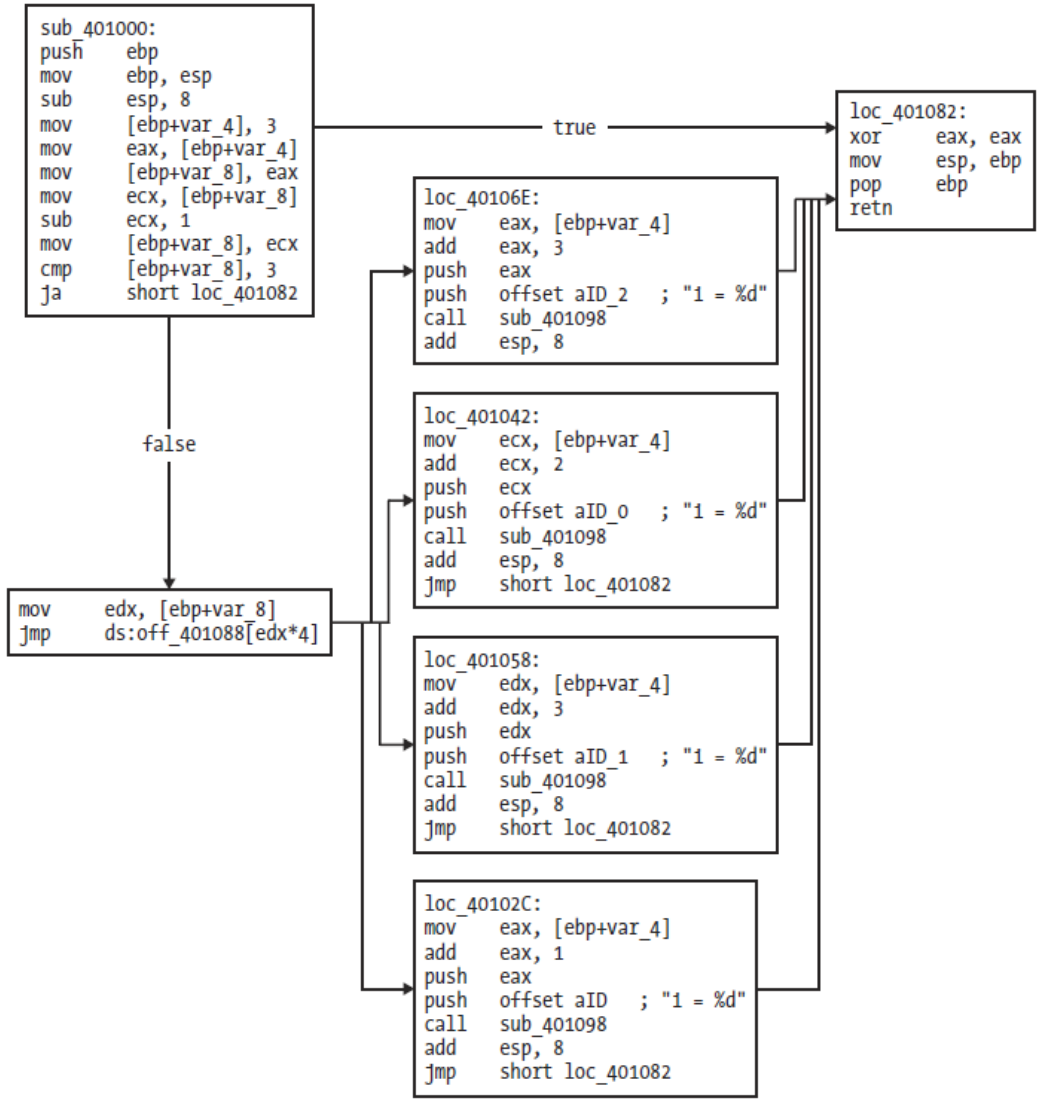
```
00401016        sub     ecx, 1
00401019        mov     [ebp+var_8], ecx
0040101C        cmp     [ebp+var_8], 3
00401020        ja      short loc_401082
00401022        mov     edx, [ebp+var_8]
00401025        jmp     ds:off_401088[edx*4] ❶
0040102C    loc_40102C:
              ...
00401040        jmp     short loc_401082
00401042    loc_401042:
              ...
00401056        jmp     short loc_401082
00401058    loc_401058:
              ...
0040106C        jmp     short loc_401082
0040106E    loc_40106E:
              ...
00401082    loc_401082:
00401082        xor     eax, eax
00401084        mov     esp, ebp
00401086        pop     ebp
00401087        retn
00401087    _main   endp
00401088 ❷off_401088  dd offset loc_40102C
0040108C              dd offset loc_401042
00401090              dd offset loc_401058
00401094              dd offset loc_40106E
```

Jump table

# Construction switch

- Jump table

```
switch(i)
{
   case 1:
      printf("i = %d", i+1);
      break;
   case 2:
      printf("i = %d", i+2);
      break;
   case 3:
      printf("i = %d", i+3);
      break;
   case 4:
      printf("i = %d", i+3);
      break;
   default:
      break;
}
```



# Tableaux

- Ils sont parcourus en utilisant une adresse de base comme point de départ et la **taille** de chaque élément peut être déterminé en analysant comment le tableau est indexé

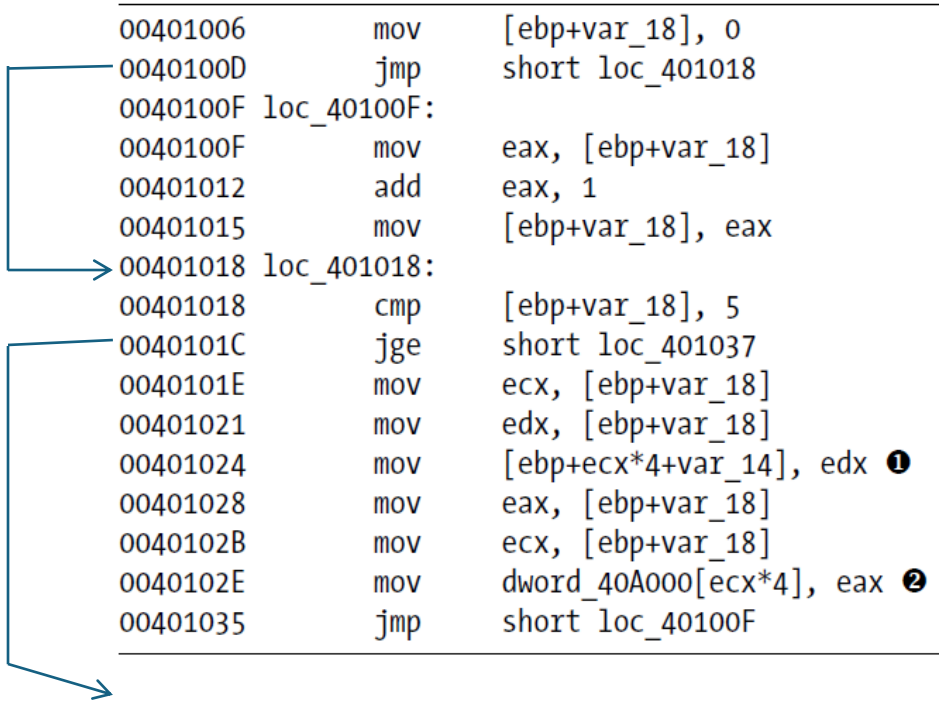
```
int b[5] = {123,87,487,7,978};
void main()
{
   int i;
   int a[5];

   for(i = 0; i<5; i++)
   {
      a[i] = i;
      b[i] = i;
   }
}
```

# Tableau

```
int b[5] = {123,87,487,7,978};
void main()
{
   int i;
   int a[5];

   for(i = 0; i<5; i++)
   {
      a[i] = i;
      b[i] = i;
   }
}
```



```
00401006         mov     [ebp+var_18], 0
0040100D         jmp     short loc_401018
0040100F loc_40100F:
0040100F         mov     eax, [ebp+var_18]
00401012         add     eax, 1
00401015         mov     [ebp+var_18], eax
00401018 loc_401018:
00401018         cmp     [ebp+var_18], 5
0040101C         jge     short loc_401037
0040101E         mov     ecx, [ebp+var_18]
00401021         mov     edx, [ebp+var_18]
00401024         mov     [ebp+ecx*4+var_14], edx ❶
00401028         mov     eax, [ebp+var_18]
0040102B         mov     ecx, [ebp+var_18]
0040102E         mov     dword_40A000[ecx*4], eax ❷
00401035         jmp     short loc_40100F
```

# Structure

- Même chose que pour les tableaux sauf que les éléments sont de types différents.

# Structure

- Même chose que pour les tableaux sauf que les éléments sont de types différents.

```
struct my_structure { ❶
     int x[5];
     char y;
     double z;
};

struct my_structure *gms; ❷

void test(struct my_structure *q)
{
     int i;
     q->y = 'a';
     q->z = 15.6;
     for(i = 0; i<5; i++){
          q->x[i] = i;
     }
}

void main()
{
     gms = (struct my_structure *) malloc(
     sizeof(struct my_structure));
     test(gms);
}
```

# Structure

- Même chose que pour les tableaux sauf que les éléments sont de types différents.
- L'adresse de la structure est passé en paramètre à la fonction

main:

```
00401050        push    ebp
00401051        mov     ebp, esp
00401053        push    20h
00401055        call    malloc
0040105A        add     esp, 4
0040105D        mov     dword_40EA30, eax
00401062        mov     eax, dword_40EA30
00401067        push    eax ❶
00401068        call    sub_401000
0040106D        add     esp, 4
00401070        xor     eax, eax
00401072        pop     ebp
00401073        retn
```

# Structure

- Même chose que pour les tableaux sauf que les éléments sont de types différents.
- Ebp+arg0  est l'adresse de base de la structure
- L'offset 0x14 sauvegarde le caractère 'a'
- Comme l'offset 0x18 (z) est un float, on utilise qword ptr
- EBP+var4 est l'adresse de i
- [ecx+eax*4]  permet d'accéder aux cases du tableau x de la structure

test:

```
00401000          push    ebp
00401001          mov     ebp, esp
00401003          push    ecx
00401004          mov     eax,[ebp+arg_0]
00401007          mov     byte ptr [eax+14h], 61h
0040100B          mov     ecx, [ebp+arg_0]
0040100E          fld     ds:dbl_40B120 ❶
00401014          fstp    qword ptr [ecx+18h]
00401017          mov     [ebp+var_4], 0
0040101E          jmp     short loc_401029
00401020 loc_401020:
00401020          mov     edx,[ebp+var_4]
00401023          add     edx, 1
00401026          mov     [ebp+var_4], edx
00401029 loc_401029:
00401029          cmp     [ebp+var_4], 5
0040102D          jge     short loc_40103D
0040102F          mov     eax,[ebp+var_4]
00401032          mov     ecx,[ebp+arg_0]
00401035          mov     edx,[ebp+var_4]
00401038          mov     [ecx+eax*4],edx ❷
0040103B          jmp     short loc_401020
0040103D loc_40103D:
0040103D          mov     esp, ebp
0040103F          pop     ebp
00401040          retn
```

# Classes et objets

```
class SimpleClass {
public:
      int x;
      void HelloWorld() {
        (1) if (  x == 10) printf("X is 10.\n");
      }
      ...
};

int _tmain(int argc, _TCHAR* argv[])
{
      SimpleClass myObject;
    ❷myObject.x = 9;
    ❸myObject.HelloWorld();
      SimpleClass myOtherObject;
      myOtherObject.x = 10;
      myOtherObject.HelloWorld();
}
```

```
;Main Function
00401100                push    ebp
00401101                mov     ebp, esp
00401103                sub     esp, 1F0h
00401109                mov     [ebp+var_10], offset off_404768
00401110              ❷ mov     [ebp+var_C], 9
00401117              ❸ lea     ecx, [ebp+var_10]
0040111A                call    sub_4115D0
0040111F                mov     [ebp+var_34], offset off_404768
00401126                mov     [ebp+var_30], 0Ah
0040112D                lea     ecx, [ebp+var_34]
00401130                call    sub_4115D0

;HelloWorld Function
004115D0                push    ebp
004115D1                mov     ebp, esp
004115D3                sub     esp, 9Ch
004115D9                push    ebx
004115DA                push    esi
004115DB                push    edi
004115DC                mov     [ebp+var_4], ecx
004115DF                mov     eax, [ebp+var_4]
004115E2          (1)   cmp     dword ptr [eax+4], 0Ah
004115E6                jnz     short loc_4115F6
004115E8                push    offset aXIs10_  ; "X is 10.\n"
004115ED                call    ds:__imp__printf
```

L'objet **this** est transmis via le registre **ecx** ou **esi**

# Fonction main

- Désassembler la fonction main
  - Sous Linux: on utilisera **gdb  filetestprogram** , puis **dis main**
  - Sous Windows, on utilisera **IDA PRO** ou **OllyDBG**
  - argv et argc se trouvent  dans la pile et sont référencés généralement par rapport à EBP

```
004113CE        cmp     [ebp+argc], 3   <--
004113D2        jz      short loc_4113D8
004113D4        xor     eax, eax
004113D6        jmp     short loc_411414
004113D8        mov     esi, esp
004113DA        push    2               ; MaxCount
004113DC        push    offset Str2     ; "-r"
004113E1        mov     eax, [ebp+argv]  <--
004113E4        mov     ecx, [eax+4]
004113E7        push    ecx             ; Str1
004113E8        call    strncmp
004113F8        test    eax, eax
004113FA        jnz     short loc_411412
004113FC        mov     esi, esp
004113FE        mov     eax, [ebp+argv]  <--
00411401        mov     ecx, [eax+8]
00411404        push    ecx             ; lpFileName
00411405        call    DeleteFileA
```