



# Génie Logiciel pour le Calcul Scientifique



#10

06/02/2025

jean-michel.batto@cea.fr

cea

[https://gogs.eldarsoft.com/M2\\_IHPS](https://gogs.eldarsoft.com/M2_IHPS)



- ❖ 1986: Rob Pike écrit un assembleur pour Plan9
- ❖ <https://9p.io/sys/doc/asm.html>
- ❖ Le flot de donnée est de gauche à droite
- ❖ L'instruction connaît la largeur du mot (Q, L)
- ❖ L'objectif de cet assembleur est la portabilité !
- ❖ <https://talks.golang.org/2016/asm.slide#1>
- ❖ Aperçu de l'assembleur :
  - ❖ FP: Frame pointer: arguments and locals.
  - ❖ PC: Program counter: jumps and branches.
  - ❖ SB: Static base pointer: global symbols.
  - ❖ SP: Stack pointer: top of stack.



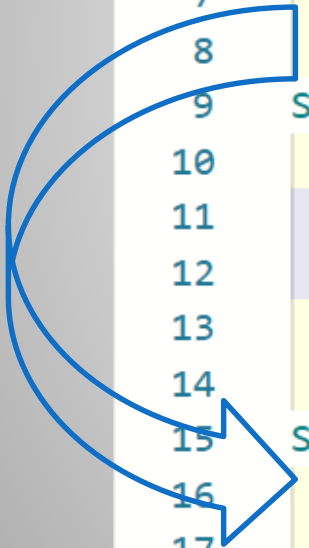
```
func Sum(a []uint64) uint64 {  
    var sum uint64  
    for i := 0; i < len(a); i++ {  
        sum += a[i]  
    }  
    return sum  
}
```

<https://godbolt.org/z/YKsW9P61T> : le code avec utilisation

# Traduction du code Go en ASM Go

amd64 gc 1.16 NOSPLIT= 4  
SP = pseudo SP, Plang symbol+offset(SP)  
 Compiler options...  
 Output... Filter... Libraries Add new... Add

1	TEXT	"" .Sum(SB), NOSPLIT ABIInternal, \$0-32		
2	FUNCDATA	\$0, gclocals·1a65e721a2ccc325b		
3	FUNCDATA	\$1, gclocals·69c1753bd5f81501d		
4	MOVQ	"" .a+16(SP), AX	SP+16 → AX → longueur SP+8 → CX → pointeur sur contenu DX : contient l'avancement BX : la somme // le résultat	
5	MOVQ	"" .a+8(SP), CX		
6	XORL	DX, DX		
7	XORL	BX, BX		
8	JMP	Sum_pc32		
9	Sum_pc16:			
10	LEAQ	1(DX), SI	LEA adresse DX+1 dans SI → taille inc CX+[DX*8] → pointeur sur contenu avec incrément Addition contenu vers BX Déplacement mémoire	
11	MOVQ	(CX)(DX*8), DI		
12	ADDQ	DI, BX		
13	MOVQ	SI, DX		
14	NOP			
15	Sum_pc32:			
16	CMPQ	DX, AX		
17	JLT	Sum_pc16		
18	MOVQ	BX, "" .~r1+32(SP)	Positionnement résultat sur la pile	
19	RET			



# Le produit scalaire – entier32, sans dépassement

→ en GoLang

```
func DotProduct(a []int32, b []int32, N int32) (sum int32) {  
    //N := len(a)  
    for i := int32(0); i < N; i++ {  
        sum += a[i] * b[i]  
    }  
    return  
}
```

→ en C

```
#include <stdint.h>  
void dp_int32(int32_t *a, int32_t *b, int32_t *len, int32_t *res)  
{  
    int32_t N = *len;  
    int32_t reslocal = 0;  
    for(int32_t i = 0; i < N; i++) {  
        reslocal = reslocal + a[i]*b[i];  
    }  
    *res = reslocal; return ; }  
.
```



- ❖ <https://godbolt.org/z/K7747xGEc>
- ❖ `-Os -mno-red-zone -mstackrealign -mllvm -inline-threshold=1000 -fno-asynchronous-unwind-tables -fno-exceptions -fno-rtti`
- ❖ `-Os` → code compact
  
- ❖ → il existe une possibilité de génération de code ASM automatique vers Golang avec :

C source #1 X

Save/Load + Add new... Vim C

```

1 // Type your code here, or load an example.
2 #include <stdint.h>
3
4 void dp_int32(int32_t *a, int32_t *b, int32_t *len, int32_t *res) {
5     int32_t N = *len;
6     int32_t reslocal = 0;
7
8     for(int32_t i = 0; i < N; i++) {
9         reslocal = reslocal + a[i]*b[i];
10    }
11    *res = reslocal;
12    return;
13 }
```

x86-64 clang 13.0.0 (C, Editor #1, Compiler #1) X

x86-64 clang 13.0.0 -Os -mno-red-zo

Output... Filter... Libraries + Add new...

```

1 dp_int32: # @d
2     push    rbp
3     mov     rbp, rsp
4     and     rsp, -8
5     mov     r8d, dword ptr [rdx]
6     test    r8d, r8d
7     jle    .LBB0_1
8     xor     eax, eax
9     xor     r9d, r9d
10
11 .LBB0_4: # =>
12     mov     edx, dword ptr [rsi + 4*rax]
13     imul   edx, dword ptr [rdi + 4*rax]
14     add     r9d, edx
15     inc     rax
16     cmp     r8, rax
17     jne    .LBB0_4
18     jmp    .LBB0_2
19
20 .LBB0_1:
21     xor     r9d, r9d
22
23 .LBB0_2:
24     mov     dword ptr [rcx], r9d
25     mov     rsp, rbp
26     pop     rbp
27     ret
```

Output (0/0) x86-64 clang 13.0.0 - 405ms (18868B) ~393 lines

- ❖ Générer un code ASM avec Clang – qui sait faire de l'autovectorisation
- ❖ Convertir le code de l'ASM Gnu vers l'ASM Go → outil c2goasm
  - ❖ git clone <https://github.com/minio/asm2plan9s>
  - ❖ git clone <https://github.com/klauspost/asmfmt>
  - ❖ git clone <https://github.com/minio/c2goasm>
  - ❖ Mettre les go.mod, mettre les binaires dans le path
  - ❖ \$ c2goasm -a -f ...



## ❖ Utilisation de c2goasm

- ❖ 1/obtenir un code C qui compile avec Clang
- ❖ `clang -S -mavx2 -masm=intel -mllvm -force-vector-width=4 -fno-asynchronous-unwind-tables -fno-exceptions -fno-omit-frame-pointer -fno-rtti -c -O3 dotproduct.c`
- ❖ 2/fabriquer un code en Go pour avoir le header
- ❖ `_dp_int32(int32_t *a, int32_t *b, int32_t *len, int32_t *res)`



```
#include <stdint.h>

void dp_int32(int32_t *a, int32_t *b, int32_t
    *len, int32_t *res) {
    int32_t N = *len;
    int32_t reslocal = 0;
    for(int32_t i = 0; i < N; i++) {
        reslocal = reslocal + a[i]*b[i];}
    *res = reslocal; return ; }
```



```
>dotproduct_amd64.go
package Dotproduct
import ( "unsafe" )
//go:noescape
func _dp_int32(a, b, N, res unsafe.Pointer)
func DotProduct(a []int32, b []int32, N int32) int32 {
    var res int32
    _dp_int32(unsafe.Pointer(&a[0]), unsafe.Pointer(&b[0]),
unsafe.Pointer(&N), unsafe.Pointer(&res))
    return res
}
```

- Ajout du '\_' en préfix
- Utilisation de unsafe
- go:noescape → pas d'analyse d'échappement



# Compilation avec Clang

- ❖ `clang -S -mavx2 -masm=intel -mllvm -force-vector-width=4 -fno-asynchronous-unwind-tables -fno-exceptions -fno-omit-frame-pointer -fno-rtti -c -O3 dotproduct.c`
- ❖ `-O3 -mavx512f -mavx512dq -mavx512bw -mavx512vbmi -mavx512vbmi2 -mavx512vl`
- ❖ Clang a 2 optimiseurs pour l'autovectorisation – on peut agir sur la taille de la vectorisation
- ❖ → génère une version assembleur
- ❖ `./c2go -a -f dotproduct.s dotproduct_amd64.s`
- ❖ Il s'agit de positionner le fichier `dotproduct_amd64.go`



- ❖ Problème : le code généré ne fonctionne pas sous windows.

→ il y a du code «en binaire»

...

```
LONG $0x646ffac5; WORD $0x1086// vmovdqu xmm4, oword [rsi + 4*rax]
```

```
LONG $0x6c6ffac5; WORD $0x2086// vmovdqu xmm5, oword [rsi + 4*rax + 16]
```

```
LONG $0x746ffac5; WORD $0x3086 // vmovdqu xmm6, oword [rsi + 4*rax + 32]
```

```
LONG $0x7c6ffac5; WORD $0x4086// vmovdqu xmm7, oword [rsi + 4*rax + 48]
```

- ❖ Autres voies explorées :

- ❖ PeachPy <https://github.com/Maratyszczka/PeachPy> - 1596 étoiles

- ❖ → assembleur codé en Python avec un métalangage

- ❖ Avo <https://github.com/mmcloughlin/avo> - 1866 étoiles

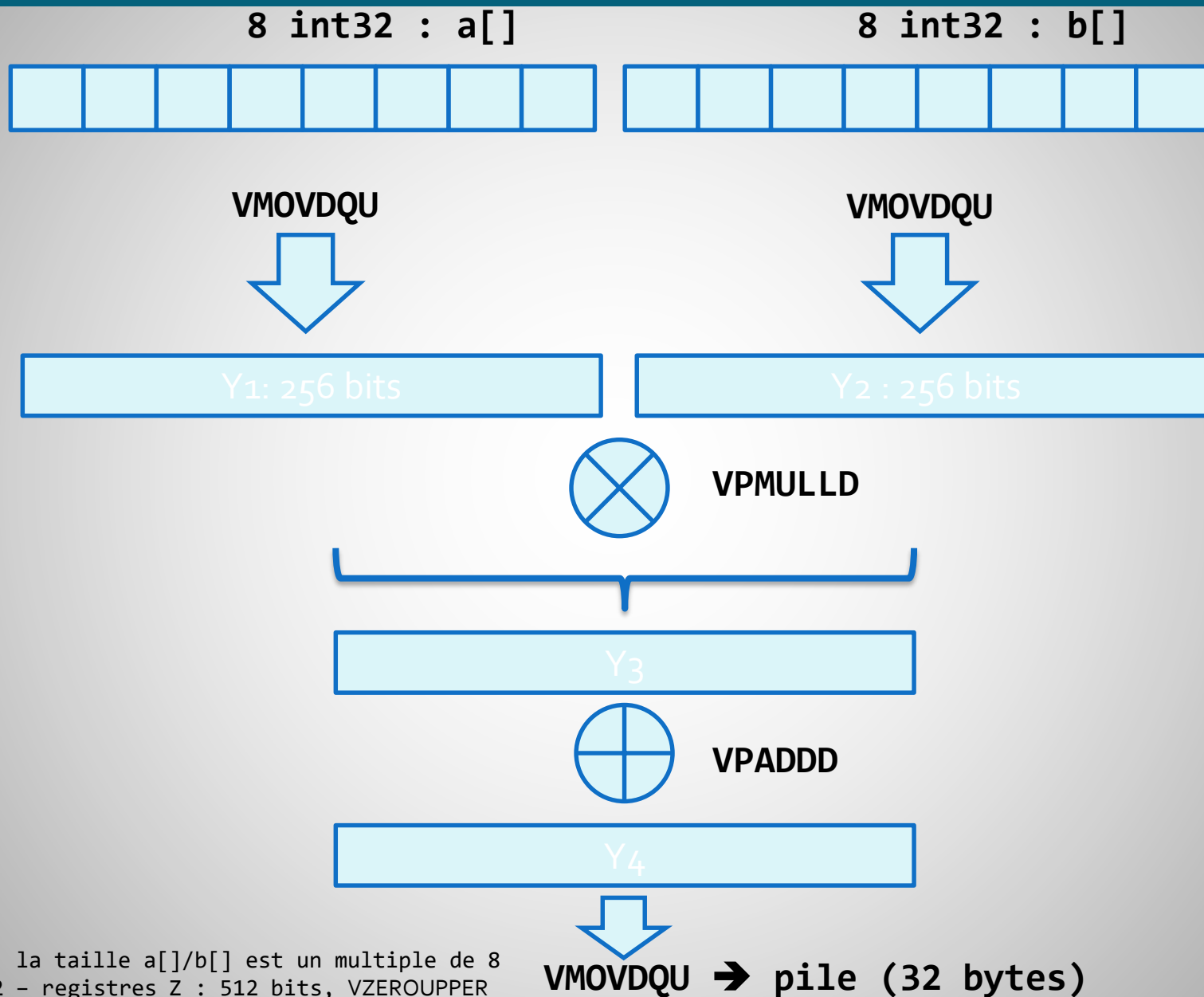
- ❖ → assembleur codé en Go avec un métalangage



# Fabrication du code 'à la main'

```
//func _dp_int32(a *int32, b *int32, gN *int32, res *int32)
TEXT    ·_dp_int32(SB),4, $0-32
        MOVQ a+0(FP), DI
        MOVQ b+8(FP), SI
        MOVQ gN+16(FP), DX
        MOVQ res+24(FP), CX
        MOVQ (DX),R8                // mov    r8d, dword [rdx]
        CMPQ R8,$0                  // test   r8d, r8d
        JLE  LBB0_1
        XORQ AX, AX                  // xor    eax, eax
        XORQ R9, R9                // xor    r9d, r9d
LBB0_4:
        MOVQ (SI)(AX*4),DX          // mov    edx, dword [rsi + 4*rax]
        IMULQ (DI)(AX*4),DX        // imul   edx, dword [rdi + 4*rax]
        ADDQ DX,R9                 // add    r9d, edx
        INCQ AX                     // inc    rax
        CMPQ AX, R8                // cmp    r8, rax
        JNE  LBB0_4
        JMP  LBB0_2
LBB0_1:
        XORQ R9, R9                // xor    r9d, r9d
LBB0_2:
        MOVQ R9,(CX)               // mov    dword [rcx], r9d
        RET
```

# Vectorisation AVX du produit scalaire



AVX : 2011

Attention : la taille `a[]/b[]` est un multiple de 8  
Avec AVX512 - registres Z : 512 bits, `VZERoupper`

`VMOVDQU` → pile (32 bytes)



```
TEXT · _dpavx_int32(SB),4, $32-32
MOVQ a+0(FP), DI
MOVQ b+8(FP), SI
MOVQ gN+16(FP), CX
MOVQ res+24(FP), DX
MOVQ (CX),R8 // value of gN
MOVQ DX,R9 // return address
VPXOR Y4, Y4, Y4
XORQ AX,AX
start:
VMOVDQU (SI), Y1
VMOVDQU (DI), Y2
VPMULLD Y1, Y2, Y3
VPADDD Y3, Y4, Y4
ADDQ $32, SI
ADDQ $32, DI
ADDQ $8, AX
CML AX, R8
JNE start
VMOVDQU Y4, d0-32(SP) // vector result to stack
LEAQ d0-32(SP), BX
MOVQ $8, CX // array length 8 int32
XORQ SI, SI // clean SI
redux: //8 bytes => 8 int32 reduction to 1 int32
ADDL (BX), SI
ADDQ $4, BX // handle 4 int32
DECQ CX
JNZ redux
MOVL SI,(R9)
RET
```

`_dpavx_int32(a *int32, b *int32, gN *int32, res *int32)`

Attention : la taille a[]/b[] est un multiple de 8





- ❖ Cible gcc/clang, SSE4.1
- ❖ Les intrinsics sont «des ajouts» au compilateur
- ❖ <https://godbolt.org/z/aW6dhrxqE>
- ❖ Utilisation du format m128 bits (XMM) // 4 int32
  - \_mm\_setzero\_si128 : mise à zéro
  - \_mm\_load\_si128 : chargement //, hint sur le cache (nocache)
  - \_mm\_mullo\_epi32 : multiplication int32, cast int32
  - \_mm\_add\_epi32 : addition int32
  - \_mm\_srli\_si128 : décalage à droite
  - \_mm\_cvtsi128\_si32 : récupère la partie basse int32



Attention : la taille a[]/b[] est un multiple de 4

```
#include <smmintrin.h>
// the actual loop body itself is still fine for runtime-variable N
[[gnu::target("sse4.1")]] int32_t dp(int32_t a[], int32_t b[], int32_t N)
{
    int32_t sum = 0;
    __m128i temp_sum = _mm_setzero_si128();
    for(int i=0;i<N;i=i+4){ // 4 int32 processed
        //Load the 4 values from x
        __m128i temp_1 = _mm_load_si128(reinterpret_cast<__m128i*>(a[i])); // add cast
        //Load the 4 values from y
        __m128i temp_2 = _mm_load_si128(reinterpret_cast<__m128i*>(b[i])); // add cast
        //Multiply x[0] and y[0], x[1] and y[1] etc
        __m128i temp_products = _mm_mullo_epi32(temp_1, temp_2);
        //Sum temp_sum
        temp_sum = _mm_add_epi32(temp_sum, temp_products);
    }
    // take horizontal sum of temp_sum - reduction
    temp_sum = _mm_add_epi32(temp_sum, _mm_srli_si128(temp_sum, 8));
    temp_sum = _mm_add_epi32(temp_sum, _mm_srli_si128(temp_sum, 4));
    sum = _mm_cvtsi128_si32(temp_sum); // convert
    // assign after the loop so compiler knows it doesn't alias
    return sum;
}
```



On évalue → `DotProductAsmAvx(a []int32, b []int32, N int32) int32`

`>go test -bench=.`

`cpu: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz`

`BenchmarkSumAsm-8 1870992 629.2 ns/op 1627.51 MB/s`

`BenchmarkDotProductAsm-8 2468762 484.7 ns/op 2112.83 MB/s`

`BenchmarkDotProduct-8 2263225 532.1 ns/op 1924.48 MB/s`

`BenchmarkDotProductAsmAvx-8 10285543 111.9 ns/op 9154.79 MB/s`



## // **creational** package pattern

// 1. **Singleton Pattern** -- une seule instance et fournir un point d'accès global à cette instance

```
type singleton struct { data string }
```

```
var ( instance *singleton once sync.Once )
```

```
func GetInstance() *singleton { once.Do(func() { instance = &singleton{data: "I am singleton"} }) return instance }
```

// 2. **Factory Method Pattern** -- interface pour la création d'un objet, mais laisse aux sous-classes le choix des classes concrètes à instancier

```
type Animal interface { Speak() string }
```

```
type Dog struct{}
```

```
type Cat struct{}
```

```
func (d *Dog) Speak() string { return "Woof!" }
```

```
func (c *Cat) Speak() string { return "Meow!" }
```

```
func CreateAnimal(animalType string) Animal {
```

```
    switch animalType {
```

```
        case "dog": return &Dog{}
```

```
        case "cat": return &Cat{}
```

```
        default: return nil } }
```



// **creational** package pattern

// 3. **Abstract Factory Pattern** -- définit une famille de classes interchangeables sans spécifier leur classe concrète

```
type Button interface { Paint() }
type WinButton struct{}
type MacButton struct{}
func (w *WinButton) Paint() { fmt.Println("Windows button") }
func (m *MacButton) Paint() { fmt.Println("Mac button") }
type GUIFactory interface {
    CreateButton() Button }
type WinFactory struct{}
type MacFactory struct{}
func (w *WinFactory) CreateButton() Button { return &WinButton{} }
func (m *MacFactory) CreateButton() Button { return &MacButton{} }
```



## // **creational** package pattern

// 4. **Builder Pattern** -- composite avec structure indépendante

```
type House struct { windows int
                    doors int
                    roof string }
type HouseBuilder struct { house *House }
func NewHouseBuilder() *HouseBuilder {
    return &HouseBuilder{house: &House{}} }
func (b *HouseBuilder) SetWindows(count int) *HouseBuilder { b.house.windows = count return b }
func (b *HouseBuilder) SetDoors(count int) *HouseBuilder { b.house.doors = count return b }
func (b *HouseBuilder) SetRoof(style string) *HouseBuilder { b.house.roof = style return b }
func (b *HouseBuilder) Build() *House { return b.house }
```

// 5. **Prototype Pattern** -- le constructeur de copie fait l'instanciation

```
type Prototype interface { Clone() Prototype }
type ConcretePrototype struct { name string }
func (p *ConcretePrototype) Clone() Prototype {
    return &ConcretePrototype{name: p.name} }
```

// **structural** package pattern

// 6. Adapter Pattern -- crée une interface pour plusieurs sous-systèmes de manière à ce qu'ils puissent interagir de manière interchangeable

```
type LegacyPrinter interface { Print(s string) string }
type MyLegacyPrinter struct{}
func (p *MyLegacyPrinter) Print(s string) string { return fmt.Sprintf("Legacy: %s", s) }
type ModernPrinter interface { PrintModern(s string) string }
type PrinterAdapter struct { LegacyPrinter }
func (p *PrinterAdapter) PrintModern(s string) string { return p.LegacyPrinter.Print(s) }
```

// 7. Bridge Pattern -- découplage

```
type DrawAPI interface { DrawCircle(x, y, radius int) }
type RedCircle struct{}
type GreenCircle struct{}
func (r *RedCircle) DrawCircle(x, y, radius int) { fmt.Printf("Drawing red circle at (%d,%d)\n", x, y) }
func (g *GreenCircle) DrawCircle(x, y, radius int) { fmt.Printf("Drawing green circle at (%d,%d)\n", x, y) }
type Shape struct { drawAPI DrawAPI }
type Circle struct { Shape x, y, radius int }
```

// 8. Composite Pattern -- structure arborescente/uniforme

```
type Component interface { Operation() string }
type Leaf struct { name string }
type Composite struct { children []Component }
func (l *Leaf) Operation() string { return l.name }
func (c *Composite) Operation() string {
    result := "Branch("
    for _, child := range c.children { result += child.Operation() + " " }
    return result + ")" }
```



// **structural** package pattern

// 9. **Decorator Pattern** -- attache des responsabilités supplémentaires à un objet de manière dynamique

```
type Coffee interface { Cost() int Description() string }
type SimpleCoffee struct{}
func (s *SimpleCoffee) Cost() int { return 5 }
func (s *SimpleCoffee) Description() string { return "Simple coffee" }
type MilkDecorator struct { coffee Coffee }
func (m *MilkDecorator) Cost() int { return m.coffee.Cost() + 2 }
func (m *MilkDecorator) Description() string { return m.coffee.Description() + ", milk" }
```

// 10. **Facade Pattern** -- regroupe plusieurs interfaces en une seule interface

```
type SubsystemA struct{} type SubsystemB struct{} type SubsystemC struct{}
func (s *SubsystemA) OperationA() string { return "Subsystem A" }
func (s *SubsystemB) OperationB() string { return "Subsystem B" }
func (s *SubsystemC) OperationC() string { return "Subsystem C" }
type Facade struct { sysA *SubsystemA
                    sysB *SubsystemB
                    sysC *SubsystemC }
func (f *Facade) Operation() string {
    return f.sysA.OperationA() + " " + f.sysB.OperationB() + " " + f.sysC.OperationC() }
```





```
// structural package pattern
```

```
// 11. Flyweight Pattern -- partage de l'état
```

```
type CharacterFlyweight struct { char rune }  
type CharacterFactory struct { chars map[rune]*CharacterFlyweight }  
func NewCharacterFactory() *CharacterFactory {  
    return &CharacterFactory{chars: make(map[rune]*CharacterFlyweight)} }  
func (f *CharacterFactory) GetCharacter(c rune) *CharacterFlyweight {  
    if f.chars[c] == nil {  
        f.chars[c] = &CharacterFlyweight{char: c} }  
    return f.chars[c] }  
// 12. Proxy Pattern -- effet miroir
```

```
type Subject interface { Request() string }  
type RealSubject struct{}  
func (s *RealSubject) Request() string { return "RealSubject: Handling request" }  
type Proxy struct { realSubject *RealSubject }  
func (p *Proxy) Request() string {  
    if p.realSubject == nil {  
        p.realSubject = &RealSubject{} }  
    return "Proxy: " + p.realSubject.Request() }  
// 13. Composite Pattern -- composition
```



```
// behaviorial package pattern
```

```
// 13. Chain of Responsibility Pattern -- répond à une demande avec découplage
```

```
type Handler interface {  
    SetNext(handler Handler)  
    Handle(request string) string }  
type AbstractHandler struct { next Handler }  
func (h *AbstractHandler) SetNext(handler Handler) { h.next = handler }  
type ConcreteHandler1 struct { AbstractHandler }  
func (h *ConcreteHandler1) Handle(request string) string {  
    if request == "one" { return "Handler1: handled" }  
    if h.next != nil { return h.next.Handle(request) } return "" }  
// 14. Command Pattern -- encapsuler une requête sous la forme d'un objet, permettant la paramétrisation des clients avec différentes requêtes
```

```
type Command interface { Execute() string }  
type Light struct{}  
func (l *Light) TurnOn() string { return "Light is on" }  
func (l *Light) TurnOff() string { return "Light is off" }  
type LightOnCommand struct { light *Light }  
func (c *LightOnCommand) Execute() string { return c.light.TurnOn() }  
// 15. Interpreter Pattern -- grammaire dans l'objet
```

```
type Expression interface { Interpret() bool }  
type TerminalExpression struct { data string }  
func (t *TerminalExpression) Interpret() bool { return len(t.data) > 0 }
```



## // behavioral

package pattern

// 16. Iterator Pattern -- se déplace sans connaître le détail de l'implémentation

```
type Iterator interface {
    HasNext() bool
    Next() interface{} }
type Collection struct {
    items []interface{} }
type CollectionIterator struct { collection *Collection index int }
func (i *CollectionIterator) HasNext() bool {
    return i.index < len(i.collection.items) }
func (i *CollectionIterator) Next() interface{} {
    if i.HasNext() { item := i.collection.items[i.index] i.index++ return item } return nil }
```

// 17. Mediator Pattern -- spécifie les interactions entre objets sans définir leurs classes concrètes

```
type Mediator interface { Notify(sender string, event string) }
type ConcreteMediator struct {
    component1 *Component1
    component2 *Component2 }
type Component1 struct { mediator Mediator }
type Component2 struct { mediator Mediator }
func (m *ConcreteMediator) Notify(sender string, event string) {
    fmt.Printf("Mediator reacts on %s and triggers event: %s\n", sender, event) }
```

## // behavioral package pattern

// 18. Memento Pattern -- informations privées pour « backup » de 1 état

```
type Memento struct { state string }
type Originator struct { state string }
func (o *Originator) CreateMemento() *Memento {
    return &Memento{state: o.state} }
func (o *Originator) RestoreMemento(m *Memento) { o.state = m.state }
```

// 19. Observer Pattern -- définit une dépendance entre objets afin qu'un changement de l'état d'un objet entraîne une mise à jour automatique

```
type Observer interface { Update(string) }
type Subject2 struct {
    observers []Observer
    state string }
func (s *Subject2) Attach(o Observer) {
    s.observers = append(s.observers, o) }
func (s *Subject2) Notify() {
    for _, observer := range s.observers { observer.Update(s.state) } }
```

// 20. State Pattern -- les états sont gérés par un handle

```
type State interface { Handle() string }
type Context struct { state State }
type ConcreteStateA struct{}
type ConcreteStateB struct{}
func (s *ConcreteStateA) Handle() string { return "State A" }
func (s *ConcreteStateB) Handle() string { return "State B" }
```



```
// behavioral package pattern
```

```
// 21. Strategy Pattern -- algorithme
```

```
type Strategy interface { Execute() string }
```

```
type ConcreteStrategyA struct{}
```

```
type ConcreteStrategyB struct{}
```

```
func (s *ConcreteStrategyA) Execute() string { return "Strategy A" }
```

```
func (s *ConcreteStrategyB) Execute() string { return "Strategy B" }
```

```
// 22. Template Method Pattern -- squelette
```

```
type AbstractClass interface {  
    TemplateMethod() string  
    PrimitiveOperation1() string  
    PrimitiveOperation2() string }
```

```
type ConcreteClass struct { PrimitiveOperation2 func() string }
```

```
type ConcreteOperation2 struct { //PrimitiveOperation2 func() string }
```

```
func (c *ConcreteClass) TemplateMethod() string { return c.PrimitiveOperation1() + " " + c.PrimitiveOperation2() }
```

```
func (c *ConcreteClass) PrimitiveOperation1() string { return "Step 1" }
```

```
func (c *ConcreteOperation2) PrimitiveOperation2() string { return "Step 2" }
```



```
// behavioral package pattern
```

```
// 23. Visitor Pattern -- pas de modification de la classe
```

```
type Visitor interface {  
    VisitConcreteElementA(*ConcreteElementA)  
    VisitConcreteElementB(*ConcreteElementB) }
```

```
type Element interface { Accept(Visitor) }
```

```
type ConcreteElementA struct{}
```

```
type ConcreteElementB struct{}
```

```
func (e *ConcreteElementA) Accept(v Visitor) { v.VisitConcreteElementA(e) }
```

```
func (e *ConcreteElementB) Accept(v Visitor) { v.VisitConcreteElementB(e) }
```

```
type ConcreteVisitor struct{}
```

```
func (v *ConcreteVisitor) VisitConcreteElementA(e *ConcreteElementA) { fmt.Println("Visited ConcreteElementA") }
```

```
func (v *ConcreteVisitor) VisitConcreteElementB(e *ConcreteElementB) { fmt.Println("Visited ConcreteElementB") }
```