LISTA 5

ZAD 1.

(data alignment)
WYRÓWNANIE DANYCH - rozmieszczenie danych w pamięci w taki sposób, aby każda z nich znajdowała się na adresie o wielokrotności jej rozmiaru.

Dlaczego dane typów prostych powinny być wyrównane? Bo dzisiejsze procesory wykonują operacje czytania i pisania po pamięci najbardziej efektywnie, gdy dane są wyrównane naturalnie; czyli gdy adres danej znajduje się na wielokrotności jej rozmiaru.

(padding)
WYPEŁNIENIE - to włożenie pustych danych między elementy tak, aby każdy element był wyrównany.

PRZYKŁAD STRUKTURY KTÓRA WAŻY WIĘCEJ NIŻ POLA

```
typedef struct{
    bool x;
    int y;
}S;
```

sizeof(x) + sizeof(y) = 5
sizeof(S) = 8

(internal padding)
WYPEŁNIENIE WEWNĘTRZNE - służy do wyrównania elementów wewnątrz struktury

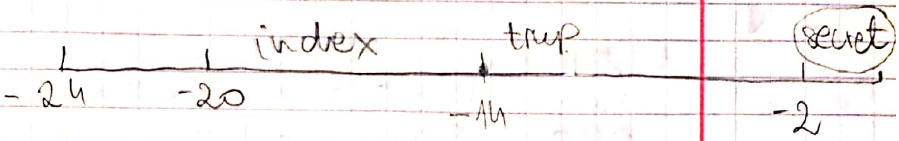
(external padding)
WYPEŁNIENIE ZEWNĘTRZNE - służy do wyrównania całej struktury (wypełnienie po ostatnim elemencie) względem największego elementu wewnątrz niej.

ZAD 2

```
int16_t you(int8_t index){
    struct{
        int16_t tbl[5];  // 2*5
        int8_t  temp;    // 1      // padding 1
        int16_t secret;  // 2
    }str;                // wielkość to 14

    str.secret = set_secret();
    //..
    return str.tbl[index];
```

```
you: pushq %rbp
movq %rsp, %rbp
subq $24, %rsp
movl %edi, %eax
movb %al, -20(%rbp)     // tu wrzucamy index
call set_secret
movw %ax, -2(%rbp)      //zapisujemy wynik set_secret
// ...
movsbl -20(%rbp), %eax  //wrzucamy index do rax
cltq
movzwl -14(%rbp,%rax,2), %eax   //zwracamy [rbp-14+rax*2]
leave
ret
```



ZATEM

Aby poznać wartość secretu powinniśmy wywołać funkcję you z argumentem 6, ponieważ sekret jest zapisany pod rbp-2, stąd

$rbp - 14 + rax*2 = rbp - 2$

$rax*2 = 12$

$rax = 6$

ZAD. 3

```
typedef struct{
    int32_t x[A][B];    //A*B*4
    int64_t y;  //8
}str1;

typedef struct{
    int8_t array[B];  // 1*B   //padding 0,1,2 lub 3
    int32_t t;        // 4   brak
    int16_t s[A];     //2*A {0,2,4,6}
    int64_t u;
}str2

void set_val(str1 *p, str2 *q){
    int64_t v1 = q->t;
    int64_t v2 = q->u;
    p->y = v1 + v2;
}

set_val:
movslq 8(%rsi), %rax     // B+ padding = 8
addq 32(%rsi), %rax      // 8+2*A+4+padding = 32
movq %rax, 184(%rdi)     // A*B*4 + {0,4} = 184
ret                                    padding
```

array -> 8
t -> 4
(12) 32bajty
2*A + padding = 20
{0,2,4,6}

B ∈ {5,6,7,8}
tablica + padding
inny

CZYLI $A*B*4 + 0 = 184$ lub $A*B*4 + 4 = 184$

gdzie $B \in \{5,6,7,8\}$

$A*B = 46$ } ale 46 nie dzieli się przez 5,6,7 ani 8

$A*B = 45$ -> podzielne przez (5)

$A*5 = 45$

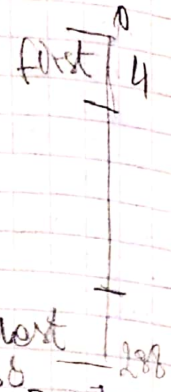
$A = 9$

a więc $A = 9$, $B = 5$

ZAD. 4.

```
typedef struct {
    int32_t first;
    a_struct a [CNT];
    int32_t last;
} b_struct;

void test (int64_t i, b_struct *bp) {
    int32_t n = bp->first + bp->last;
    a_struct *ap = &bp->a[i];
    ap->x [ap->idx] = n;
}
```



40 to rozmiar struktury A

wielkość a [CNT]

$16^2 + 16 \cdot 2 = 288$

```
test:
    movl  0x120(%rsi), %ecx            //ecx = bp.last
    addl  (%rsi), %ecx                 //ecx (n) = bp.first + bp.last
    leaq  (%rdi, %rdi, 4), %rax        //rax = 5i
    leaq  (%rsi, %rax, 8), %rax        //rax = bp + 40i (adres)      bp->a[i]
    movq  0x8(%rax), %rdx              //rax = bp + 40i + 8 (wartość)
    movslq %ecx, %rcx                                    - int32_t + padd
    movq  %rcx, 0x10(%rax, %rdx, 8)    //bp + 16 + 40i + 8*idx
                                       //bp + 40i + 8 + 8*idx + 8
                                       //bp->a[i] + 8*idx + 8
                                       //bp->a[i]->x[idx]
    retq
```

rax = bp + 4(i); size(A) = 40 ←

wiedząc, że se padding kończy się w bp + 8

→ Zatem A * CNT = 288 − 8 (first), czyli CNT = 280/40 = 7

CNT = 7

```
typedef struct {
    int64_t idx;     → 8 bajtów
    int64_t x[4];    } zostaje 32 bajty, a 32/8 = 4
} a_struct;
```

ZAD. 5

```
int32_t cread (int32_t *xp) {
    return (xp ? *xp : 0);
}
```

jeżeli xp != 0 to zwróć to na co wskazuje (wartość), wpw. zwróć 0

Semantyka cond ? x : y oznacza

```
if (cond) return x;
else return y;
```

~~Podany na liście kod nie działa poprawnie – xp zastosowane na bool zawsze daje true, należy użyć xp = NULL~~

Chcemy przenieść wartość jeśli poprzednia instrukcja ustawiła flagę ZF

```
#include <iostream>

int cread(int *xp) {
    int defaultVal = 0;
    int *yp = &defaultVal;
    int *res = (xp != NULL) ? xp : yp;
    return *res;
}
```

→ optymalizacja kodu

wskaźnik na 0

czy jest wskaźnikiem

ZAD 8. A [SIZE1][SIZE2] ->

W asemblerze tablice wielowymiarowe są w pamięci przechowywane jako tablice jednowymiarowe

1	2	3	...	SIZE 1
SIZE2	SIZE2	SIZE2		SIZE2

ZAD. 4 Pole bitowe - 4 końcowe bity / n bitów końcowych

int U2

10 -> 1010 -> -8+0+2+0 = -6

uint

10 -> 1010 -> 10

sal shift lewy zwykły

sarb shift prawy arytmetyczny

cl -> 8 bitów 1111,1010

Pole bitowe typu int o rozmiarze 4 - zamiast całej wartości bierzemy 4 ostatnie bity, czyli przykładowo w liczbie 100100.11 interesuje nas 00.11

```
void set_field(void){
struct{
    int32_t field:4; // wzięcie pola bitowego o rozmiarze 4
}s;
  s.field = 10
  printf("Field value is:" %d \n", s.field);
```