Eksamen var en moodle quiz, så her er spørgsmålene med svarene blanket ud i tilfælde af at du gerne vil prøve at lave den uden at have svarene.

Betragt den (hypothetiske) CAOSv1 maskine som anvender **w=5-bit words** til at repræsentere heltal. Det er givet at maskinen anvender two's complement til at repræsentere tal med fortegn (signed). Betragt nedenstående erklæringer i C.

```
int x1 = INT_MIN; //TMin_w
int x2 = INT_MAX; //TMax_w
int x3 = 2 ;
unsigned int y1 = UINT_MIN;
unsigned int y2 = UINT_MAX;//UMax_w
```

Question 1

Udfyld nedenstående tabel

Svar

Nr.	Udtryk	Værdi (decimal)
1	x1	[redacted]
2	x2	[redacted]
3	y1	[redacted]
4	y2	[redacted]

Udfyld nedenstående tabel.

I delopgave 9 og 10 skal du bruge værdierne 0 for falsk og 1 for sand.

Svar

Nr	Udtryk	Beregnes som?	Overflow?	Værdi (decimal)
5	x3+x1	[redacted]	[redacted]	[redacted]
6	x3+x2	[redacted]	[redacted]	[redacted]
7	y1-x3	[redacted]	[redacted]	[redacted]
8	y1+x3	[redacted]	[redacted]	[redacted]
9	(x1-x3<0)	[redacted]	[redacted]	[redacted]
10	(x1+x2<0)	[redacted]	[redacted]	[redacted]

Svarmulighederne var
Beregnes som? Signed, unsigned eller ingen

Overflow? Ingen overflow, Overflow i positiv retning eller overflow i negativ retning

Question 3

Complete

Marked out of 5.00

Flag question

Lad x være et 32-bit ord erklæret i C som

```
unsigned int x;
```

Hvilke blandt nedenstående udtryk udtrækker værdien af den mest betydende byte i x?

Select one or more:

a. (x&0xff000000)>>24

b. (x&0xff00)>>8

c. x/0x1000000

d. x<<8

e. x|0x000000ff

f. x>>24

Question 4

Complete

Marked out of 5.00

Flag question

Hvilke af nedenstående udsagn er **SANDE?**

Select one or more:

a. Instruktionen leaq udlæser 8 bytes fra hukommelsen

b. Instruktionen movq udlæser 4 bytes fra hukommelsen

c. Indholdet af stackpointer registeret %rsp gemmes den kaldte (callee saved)

d. Instruktionen call dekrementerer stackpointer (%rsp)

e. Hvis %rax indholder 0, og %rdx indeholder 8, så vil instruktionen addq %rdx,%rax sætte sign-flag til 1

I det følgende er vist et fragment af et X86-64 assembly program, som gcc har oversat fra et lille C-program. Ligeledes er vist et fragment af hukommelsen.

```
0000000000401136 <func>:
01149 <func>:
    1149: movq    (%rdi),%rax
    114c: movq    (%rsi),%rdx
    114f: cmpq    %rdx,%rax
    1152: jg     115a
    1154: movq    $0x0,%eax
    1159: retq
    115a: movq    %rdx,(%rdi)
    115d: movq    %rax,(%rsi)
    1160: movq    $0x1,%eax
    1165: retq

01166 <do_func>:
    1166: leaq    0x8(%rdi),%rsi
    116a: callq  1149
    116f: retq
```

Hukommelsen

Adresse 0x..	Værdi
679040	4
679048	3
679050	2
679058	1

Question 5 Complete Marked out of 3.00 Flag question

Hvilke parameter typer anvender programmets funktioner?

Hvilken type har func's parameter 1 ? [redacted]

Hvilken type har func's parameter 2 ? [redacted]

Hvilken type har do_func's parameter 1? [redacted]

Svarmulighederne var datatyper, såsom char, short, long, char* osv.

Hvilken værdi returnerer **do_func(0x679040)** (svar i hex, uden 0x prefix)? 0x...

Answer:

Question 7

Complete

Marked out of 8.00

Flag question

Lav et beregningsspor (execution trace) af fragmentet ved at udfylde nedenstående tabel givet start tilstanden givet i første linie. Fragmentet er starter med kald til **do_func**(0x679040).

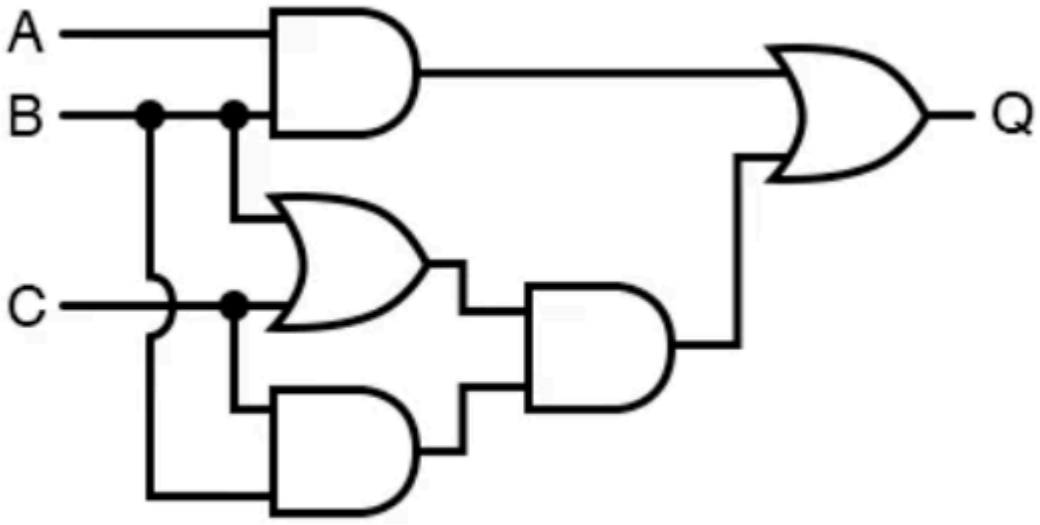
Dvs. angiv effekten på processorens tilstanden ved at hånd-eksekvere instruktionerne én efter én. I linie n skal du angive den tilstand, der gælder inden instruktionen udføres; dens resultat (ændring af register-værdier) skal dermed stå i linien under: n+1. Hvis en værdi ikke kan udledes, bedes du angive dette med et "-". Stands når/hvis eksekvering af **do_func** når "ret".

Angiv alle tal i hex (uden 0x-prefix, som her er underforstået).

Beregningsspor

se-kvens	PC (%rip) 0x... : instruktionsnavn	%rsp 0x...	%rax 0x...	SF
0	1166: leaq 0x8(%rdi),%rsi	7ffd9d0	-	-
1	[redacted]	[redacted]	-	-
2	[redacted]	[redacted]	[redacted]	[redacted]
3	[redacted]	[redacted]	[redacted]	[redacted]
4	[redacted]	[redacted]	[redacted]	[redacted]
5	[redacted]	[redacted]	[redacted]	[redacted]
6	[redacted]	[redacted]	[redacted]	[redacted]
7	[redacted]	[redacted]	[redacted]	[redacted]
8	[redacted]	[redacted]	[redacted]	[redacted]
9	[redacted]	[redacted]	[redacted]	[redacted]
10	[redacted]	-	-	-

Spørgsmålet omhandler kombinatoriske kredsløb og boolsk algebra.
Betragt nedenstående kombinatoriske kredsløb:



Hvilke af nedenstående boolske udtryk er ***ækvivalente*** (giver samme resultat/output) med kredsløbets output Q?

(her anvendes notationen for boolske operatorer som vist på side 87 i lærerbogen: & | ~ ^

Select one or more:

a. `A&B | B&C&(B|C)`

b. `A&B | B&C & B&C`

c. `A&B&C`

d. `B&(A|C)`

I denne opgave antager vi processor arkitekturen SEQ Y86-64, som beskrevet i lærebogen.

Spor effekten af udførelsen af den konkrete instruktion ved at udfylde nedenstående tabel med specifikke konkrete **hexadecimale værdier** for instruktionen **call 0x77774402.** Det er givet, at **%rsp** har værdien **0x4458** og at program tælleren PC har værdien **0x16aff10**. Alle værdier er underforstået HEX, undlad derfor 0x prefix.

Stage	CALL *dst*
Fetch	icode:ifun ← M_1[PC] valC ← M_8[PC+1] valP ← PC+9
Decode	valB←R[%rsp]
Execute	valE ← valB+(-8)
Memory	M_8[valE] ← valP
Write back	R[%rsp]← valE
PC update	PC ← valC

Stage	CALL 0x77774402
Fetch	icode:ifun ← M_1[...]=... : ... valC ← ... =... valP ← ... = ...
Decode	valB ←...=...
Execute	valE ← ...=...
Memory	... ← ...
Write back	... ← ...
PC update	PC ← ...

Eksekvering (undlad 0x prefix)

Trin	**call 0x77774402**
Fetch	icode [ ] : ifun [ ] <- M1[[ ]] valC <- M8[[ ]] valP <- [ ]
Decode	valB <- [ ] <- R[[ ]]
Execute	valE <- [ ] <- valB + (-8)
Memory	M8[[ ]] <- [ ]
Writeback	R[[ ]] <- [ ]
PC update	PC <- [ ]

Information

Flag question

Program Optimering og Instruktionsniveau Parallelitet (15 pts)

Nedenstående listing 1 viser en simpel dynamisk kædet liste i C. Listen består af knuder (node_t), der er hægtet sammen vha. pointere, og kan gemme et enkelt heltals-værdi (element). Datatypen list_t gemmer listens hoved og hale. Listen her understøtter 2 operationer: at tilføje nye elementer til halen af listen (append), og summere værdien af alle elementer i listen (sumList). Koden anvender malloc, som dynamisk afsætter de krævede antal bytes i processens hob-hukommelse et eller andet sted hvorend der måtte være plads.

Listing 2 indeholder en anden variant af en kædet liste. Den primære forskel er, at hver knude nu kan gemme flere (COUNT) elementer. Antal brugte element pladser styres af knudes used-variabel.

Analyser de 2 varianters adgangsmønster til hukommelsen, med særlig fokus på append, main, og sumList.

Listing 1: En kædet liste i C.

```c
#include<stdio.h>
#include<malloc.h>

typedef struct node {
  int elem;            //værdien gemt i liste-knuden
  struct node * next; //pointer til næste knude i listen
} node_t;

typedef struct {         //samler listens hoved og hale.
  node_t * head;
  node_t * tail;
} list_t;

node_t * makeNode(){       //alloker og initialiser ny knude
  node_t * n=(node_t*) malloc(sizeof(node_t));
  n->next=NULL;
  return n;
}
void append(list_t* l, int elem){
  node_t * n=makeNode(); //lav plads til nyt element
  if(l->head==NULL)        //tom liste?
    l->head=n;             //indsaet ny hoved
  else
    l->tail->next=n;       //indsaet knude i halen af listen
  l->tail=n;               //opdater hale-knude så den peger på nyt knude
  n->elem=elem;            //indskriv elementets værdi
}
```

```c

void initList(list_t *l){
  l->head=NULL; l->tail=NULL;
}

int sumList(list_t *l){ //itererer listen og summerer værdierne
  int sum=0;
  node_t * e=l->head;    //start ved hoved
  while(e!=NULL){        //nået enden?
    sum+=e->elem;
    e=e->next;           //hent næste element
  }
  return sum;
}

int main(){
  list_t myList;
  initList(&myList);

  for(int i=0;i<100;i++)  append(&myList,i);
  printf("SUM: %d\n",sumList(&myList));
}
```

Listing 2: Variant 2 af en kædet list i C.

```c
#include<stdio.h>
#include<malloc.h>
#define COUNT (64/sizeof(int))

typedef struct node_v2 {
  int elems[COUNT];       //der er plads til COUNT elementer i en knude
  int used;               //af dem er used pladser brugte
  struct node_v2 * next; //næste knude i listen
} node_v2_t;

typedef struct {
  node_v2_t * head;
  node_v2_t * tail;
} list_v2_t;

node_v2_t * makeNode_v2(){
  node_v2_t * n=(node_v2_t*) malloc(sizeof(node_v2_t));
  n->used=0; n->next=NULL;
  return n;
}
void initList_v2(list_v2_t *l){
  l->head=NULL; l->tail=NULL;
}

void append_v2(list_v2_t*l, int elem){//alloker og initialiser ny knude
  if(l->head==NULL) {                  //tom liste?
    node_v2_t * first=makeNode_v2();
    l->head=first;l->tail=first;       //indsæt første knude
  }

  if(l->tail->used>=COUNT) {           //all pladser i hale-knuden brugt?
    node_v2_t * new=makeNode_v2();
    l->tail->next=new; l->tail=new;
  }
  l->tail->elems[l->tail->used]=elem; //indsæt elementet
  l->tail->used++;
}
int sumList_v2(list_v2_t *l){
  int sum=0;
  node_v2_t * n=l->head;               //start ved hoved
  while(n!=NULL){                      //nået enden?
    for(int i=0;i<n->used;i++)         //summer elementerne i knuden
      sum+=n->elems[i];
    n=n->next;                         //udpeg næste knude
  }
 return sum;
}
int main(){
  list_v2_t myList_v2;
  initList_v2(&myList_v2);
  for(int i=0;i<100;i++)  append_v2(&myList_v2,i);
  printf("SUM: %d\n",sumList_v2(&myList_v2));
}
```

Hvilken variant udviser bedst spatial lokalitet?

Select one:

a. Variant 2 har bedst spatial lokalitet

b. Da de 2 varianter begge benytter pointere har de lige ringe lokalitet

c. Variant 1 har bedst spatial lokalitet

d. Da de 2 varianter begge bruge stucts har de lige god lokalitet

Betragt append_v2 og kaldene dertil i main. Hvilke af funktionens variable bliver brugt med god temporal lokalitet?

Select one or more:

a. l->tail

b. first

c. l->head

d. l->tail->elems[l->tail->used]

e. l->tail->used

f. l->tail->next

Question 12

Complete

Marked out of 5.00

Flag question

Listing 3 indeholder et forsøg på optimering af sumList_v2.

Listing 3: Et yderligere forsøg på optimering af funktion.

```
1
2  int sumList_v3(list_v2_t *l){
3    int sum=0;
4    int tmp=0;
5    int i=0;
6    node_v2_t * n=l->head;
7    while(n!=NULL){
8      int limit=n->used-1;
9      tmp=0;
10     for(i=0;i<limit;i+=2){
11       sum+=n->elems[i];
12       tmp+=n->elems[i+1];
13     }
14     sum+=tmp;
15     for(;i<n->used;i++) sum+=n->elems[i];
16       n=n->next;
17   }
18   return sum;
19 }
```

Hvilke optimeringer er anvendt i sumList_v3 i forhold til sumList_v2?

Select one or more:

Hvilke optimeringer er anvendt i sumList_v3 i forhold til sumList_v2?

Select one or more:

a. 2x1 unrolling (2 gange løkkeudfoldning med 1 akkumulator variabel)

b. elimineret optimeringsblokkeren procedurekald

c. 1x1 unrolling (1 gange løkkeudfoldning med 1 akkumulator variabel)

d. 2x2 unrolling (2 gange løkkeudfoldning med 2 akkumulator variable)

e. elimineret race-conditions i processorens pipeline

f. 2x3 unrolling (2 gange løkkeudfoldning med 3 akkumulator variable)

Hukommelses Organisering (15 pts)

I denne opgave skal du oversætte virtuelle til fysiske adresser i nedenstående mini hukommelsesystem givet ved:

- Sidestørrelsen er 128 bytes
- Virtuelle adresser er 14 bits lange
- Fysiske adresser er 12 bits lange
- Der er tilknyttet en 3-vejs TLB (translation lookaside buffer) med 4 sets. Hukommelsen kan udlæses byte-vist.
- Sidetabel og TLB har følgende indhold (alle hexa-decimale værdier):

VPN	PPN	Valid
28	06	0
29	17	1
2A	18	0
2B	09	1
2C	0A	1
2D	1B	1
2E	01	1
2F	00	0

VPN	PPN	Valid
40	1F	1
41	0E	1
42	0D	1
43	0C	1
44	13	1
45	12	1
46	19	1
47	08	1

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	10	13	1	0B	0A	1	0A	0F	0
1	0A	19	1	0B	03	1	10	0E	1
2	11	15	0	10	19	1	0A	01	0
3	10	0C	1	0A	11	0	0B	00	0

Svarmulighederne var “Side offset” , “TLB-Cache Index” og “TLB-Cache Tag”

Vis hvordan den virtuelle adresse **0x17aa** oversættes til fysisk adresse.
Brug minus symbolet "-" til at angive at en værdi ikke kan genereres.
Angiv værdier i HEX (undlad 0x-prefix).

Adresse oversættelse

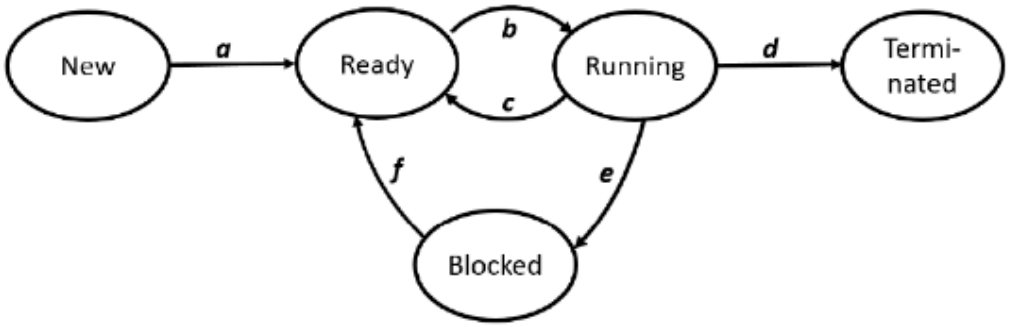
Parameter	Værdi (hex)
VPN (0x...):	
TLB Indeks (0x...):	
TLB-Tag (0x...):	
TLB-Hit (j/n):	
Page fault (j/n):	
PPN (0x...):	

Vis hvordan den virtuelle adresse **0x2201** oversættes til fysisk adresse. Brug minus symbolet "-" til at angive at en værdi ikke kan genereres. Angiv værdier i HEX (undlad 0x-prefix).

Adresse oversættelse

Parameter	Værdi (hex)
VPN (0x...):	
TLB Indeks (0x...):	
TLB-Tag (0x...):	
TLB-Hit (j/n):	
Page fault (j/n):	
PPN (0x...):	

Betragt trådtilstandsmodellen i nedenstående figur.



Angiv med bogstaverne a - f den tilstandsovergang, der resulterer af nedenstående situationer:

En tråd signaleres via cond_signal på en condition variabel

Tråden har sendt et HTTP request og skal nu afvente et HTTP respons

Trådens stak og kontrol-blok er allokeret og initialiseret

En kørende tråds kerne skal bruges af af en anden tråd med højere prioritet

Tråden kalder cond_wait på en condition variabel

Betragt følgende C/Unix program.

```c
int x=1;
int status;
pid_t pid;

int main ( ) {
  pid=fork() ;
   if (pid==0){
       pid=fork();
       if(pid==0) {
         x+=2;
         printf ("A %d\n" , x ) ;
       }
       else {
       wait(&status);
       printf ("B %d\n" , x ) ;
     }
     printf ("C %d\n" , x ) ;
  }
  else {
   wait (&status) ;
   printf ("D %d\n" , x ) ;
  }
  exit (0) ;
}
```

Giv et eksempel på hvad programmet kan skrive ud (antag her at printf udføres atomisk).

Udskrift:

linie 1:

linie 2:

linie 3:

linie 4:

linie 5:

A 1 | A 3 | B 1 | B 3 | C 1 | C 3

D 1 | D 3 | Tom Linie

Betragt igen ovenstående C/Unix program. Antag her at printf afvikles atomisk. Hvor mange forskellige outputs kan programmet generere ?

Answer:

Sidste opgave er en programmerings opgave
Denne opgave omhandler concurrency og flertrådet programmering med pthreads.

Målet er at lave parallel søgning efter løsninger med en (tilstrækkelig) god "pris" (cost): Prisen kan fx være distancen for en rute til en mål-knude i en stor vægtet graf. Selve søgningen er ikke en del af opgaven, men er blot simuleret ved en tilfældig-tal generator, som giver en "løsning" med en tilfældig pris/cost.

Nedenstående program-skellet opretter et antal tråde, som er "søge-agenter" (searchAgent). Når en søgeagent finder en ny løsning, sammenligner den prisen på den nye løsning (solutionCost) med den bedste pris(bestCost), som de hidtil har fundet. Hvis den nye løsning er billigere opdateres den hidtil bedste pris med den nye (i funktionen updateCost). En søge-agent terminerer når den har fundet 10 løsninger (agentDone).

Ligeledes opretter programmet 2 brugere (user), som efterspørger en løsning på en højest ønsket pris(requiredCost), i eksemplet priser på hhv. 100 og 2000. En bruger-tråd skal afvente indtil en løsning er fundet med en bedre pris end den forespurgte, eller til søgeagenterne har termineret (hvad end der kommer først).

Færdiggør nedenstående program-skellet med korrekt gensidig udelukkelse og synkronisering vha. pthreads locks og condition variable. Specifikt, 1) lav de nødvendige ekstra erklæringer, lav ny udgave af funktionerne: 2) updateCost, 3) awaitCost), og 4) agentDone.

Hint: Hvis du ikke kan eksakt C-syntaks, kan du prøve med plausibel pseudo kode.

INDSÆT DE EKSTRA GLOBALE ERKLÆRINGER OG OPDATEREDE FUNKTIONER NEDENFOR.

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<unistd.h>
#include<limits.h>
#include"common_threads.h"

#define noAgents 3

//INITIALIALIZATIONS
int agentsDone=0;     //number of agents that have finished searching
int bestCost=INT_MAX; //int_max means nothing found

int search(){
  sleep(1);    /* simulates computing time searching for a goal in a (large) weighted graph*/
  int solutionCost = rand() % 10000; //random valued solution cost between 0 an 10000
  return solutionCost;
```

```
}

void updateCost(int solutionCost){
  if(solutionCost<bestCost) {
    bestCost=solutionCost;
  }
}
void agentDone(){
  agentsDone++;
}

int awaitCost(int requiredCost){
  int foundCost;
  //wait if the current best cost exceeds required cost or some agents are still working
  if (bestCost>requiredCost) ; //wait
  if (agentsDone<noAgents) ; //wait
  foundCost=bestCost;

  return foundCost;
}

void * searchAgent(void *arg){
  for(int i=0;i<10;i++){        //find 10 solutions
    int solutionCost=search();
    updateCost(solutionCost);   //and update bestCost
  }
  agentDone();                  //declare agent is done
}

void * user(void *arg){         //arg contains expected csst
  int cost=awaitCost((int)arg); //cast will give compiler warning: ignore
  printf("Found solution at cost %d\n",cost);
}

int main(int argc,char argv[]){
srand((unsigned)time(NULL));   //seed pseudo random generator

 pthread_t agent[noAgents];    //start threads
 pthread_t user1,user2;
 for (int i=0;i<noAgents;i++)
   Pthread_create(&agent[i],NULL,searchAgent,(void*)i);

 Pthread_create(&user1,NULL,user,(void*)100);
 Pthread_create(&user2,NULL,user,(void*)2000);
                            //await termination
 Pthread_join(user1,NULL); Pthread_join(user2,NULL);
 for (int i=0;i<noAgents;i++)
   Pthread_join(agent[i],NULL);
```

```
 printf("All Done %d\n",bestCost);
 return 0;
}
```

Answer text Question 19
Erklæringer

```
pthread_mutex_t mutex = PTHREAD_MUTED_INITIALIZER;
pthread_cond_t waiters = PTHREAD_COND_INITIALIZER;
int agentsDone= 0;  //En global variable tæller de ankomne

//Et par steder bør jeg indsætte pthread_cond_(wait | broadcast) for at afvente svar og
singalere svar
//jeg kommer igen senere hvis jeg har tid


void updateCost(int solutionCost){
  if(solutionCost<bestCost) {
    bestCost=solutionCost;
  }
}

void agentDone(){
  Pthread_mutex_lock(&lock);
  agentsDone++;            //Vigtigt at denne låses så der ikke kludres i værdien!
  Pthread_mutex_lock(&unlock);
}

int awaitCost(int requiredCost){
  int foundCost;
  //wait if the current best cost exceeds required cost or some agents are still working
  if (bestCost>requiredCost) ; //wait
   pthread_cond_(wait)
  if (agentsDone<noAgents) ; //wait
   pthread_cond_(wait)
  foundCost=bestCost;
pthread_cond_(broadcast) //Tror den her er malplaceret med har ikke lige tiden til at blive
sikker og rette

  return foundCost;
}
```