Tema 3 - Infernul lui Biju

- Deadline: 23.01.2021
- Data publicării: 13.12.2020
- Responsabili:
  - Andrei Toma [mailto:andrei.toma1009@stud.acs.upb.ro]
  - Marian Burcea [mailto:mariangabriel057@gmail.com]
  - Alexandru Ilie [mailto:alexandru.ilie2108@stud.acs.upb.ro]
  - Stefan-Theodor Ionica [mailto:stefan.ionica@stud.acs.upb.ro]
  - Cristian Vijelie [mailto:cristianvijelie@gmail.com]

Ultima actualizare checker: 04 ianuarie 2022, 19:25; a fost adaugat task-ul bonus de instructiuni vectoriale.

Ultima actualizare enunt: 04 ianuarie 2022, 19:25; a fost adaugat task-ul bonus de instructiuni vectoriale

Enunț

Biju a murit de inima rezolvand Tema 2, asa ca a ajuns la judecata barosanilor programarii, unde i se va verifica valoarea. Barosanii ii arata mai multe usi, in spatele carora se afla cate un calculator care stie doar Assembly, si un greu al programarii, care il va observa cu mare atentie in timp ce rezolva provocarile. I se mai spune ca singura conditie in care se poate intoarce pe Pamant este sa ia 100 de puncte sau mai mult. De asemenea, daca va lipsi prea mult de pe Pamant, va fi uitat de ceilalti muritori si nu va mai fi lasat sa plece.

Aceasta tema este structurata in mai multe task-uri independente, dintre care 4 fac parte din tema actuala, iar celelalte sunt bonusuri. Task-urile bonus vor avea cerintele fiecare in folder-ul aferent, in repository-ul de Github

Task-ul 1 - Sortarea albumelor (25p)

Intrand pe prima usa, Biju este intampinat de Allen Newell, unul din inventatorii listelor inlantuite. Acesta il provoaca pe Biju sa sorteze albumele unuia dintre rivalii lui, Bogdan de la Ploiesti. Albumele lui, fiind pretioase, sunt tinute in niste noduri ale unei liste simplu-inlantuite.

Biju trebuie să implementeze funcția cu semnătura `struct node* sort(int n, struct node* node);` din fișierul task1.asm, care "leagă" nodurile din listă în ordine crescătoare. Funcția primește numărul de noduri și adresa vectorului și întoarce adresa primului nod din lista rezultată.

Structura unui nod este:

```
struct node {
    int val;
    struct node* next;
};
```

și, inițial, câmpul **next** este setat la **NULL**.

Precizări:

- n >= 1
- secvența conține numere consecutive distincte începand cu 1 (ex: 1 2 3 …)
- structura vectorului NU trebuie modificată (interschimbarea nodurilor este interzisă)
- sortarea trebuie facută in-place
- este permisă folosirea unor structuri auxiliare, atâta timp cât, nodurile listei rezultate sunt cele din vectorul inițial
- NU este permisă folosirea funcției **qsort** din libc
- NU este permisă folosirea variabilelor globale

Exemplu

Inițial:

Adresă	Valoare	Next
0x32	2	NULL
0x3A	1	NULL
0x42	3	NULL

Apelul funcției **sort** întoarce **0x3A** (adresa nodului cu valoarea 1) și vectorul va arăta astfel:

Adresă	Valoare	Next
0x32	2	0x42
0x3A	1	0x32
0x42	3	NULL

Pentru implementarea sortării vă puteți inspira din Selection Sort [https://www.geeksforgeeks.org/selection-sort/]

Punctare

Task-ul are 25 de puncte, dintre care un punct pentru coding-style si detalierea implementarii.

Task-ul 2 - Cosmarul lui Turing (25p)

Dupa a 2-a usa, Biju este intampinat de Alan Turing. Acesta ii pune in fata o Masina cu stiva [https://en.wikipedia.org/wiki/Stack_machine], adica o masina care stie sa foloseasca doar stiva, prin instructiuni de tip **push** si **pop**, pentru a lucra cu memoria. Provocarea lui biju este sa implementeze 2 functii pe aceasta masina:

CMMMC

Prima functie este `int cmmmc(int a, int b)`, care calculeaza cel mai mic multiplu comun a 2 numere, date ca parametru.

Se garanteaza ca rezultatul inmultirii lui a si b incape pe 4 bytes

Paranteze

A doua functie este `int par(int str_length, char *str)`, care verifica daca o secventa de paranteze este corecta. Aceasta primeste un sir catre contine **doar paranteze rotunde** si lungimea sirului, si intoarce 1, daca secventa e corecta, sau 0, daca secventa e gresita.

Exemplu

- Pentru secventa ((()())(())), rezultatul va fi 1
- Pentru secventa (())((, rezultatul va fi 0

Punctare

Task-ul are 25 de puncte, dintre care 2 puncte pentru coding-style si detalierea implementarii.

Task-ul 3 - Sortare de cuvinte (25p)

Dupa a 3-a usa, Biju e intampinat de John von Neumann, inventatorul Merge Sort, printre altele. John ii spune lui Biju ca are probleme in a intelege versurile lui Salam, si ii cere sa scrie 2 functii care sa-l ajute in descifrarea lor.

Pentru acest task veti avea de separat un text in cuvinte dupa niste delimitatori si, dupa aceea, sa sortati aceste cuvinte folosind functia qsort. Sortarea se va face intai dupa lungimea cuvintelor si in cazul egalitatii se va sorta lexicografic.

Va trebui sa implementati 2 functii cu semnaturile `void get_words(char *s, char **words, int number_of_words);` si `void sort(char **words, int number_of_words, int size);` din fisierul task3.asm. Functia get_words primeste ca parametri textul, un vector de stringuri in care va trebui sa salvati cuvintele pe care ulterior le veti sorta si numarul de cuvinte. Functia sort va primi vectorul de cuvinte, numarul de cuvinte si dimensiunea unui cuvant.

Scopul task-ului acesta este sa folositi functii din biblioteca standard C. Daca exista ceva in libc ce va poate ajuta, folositi cu incredere!

Precizări

- lungimea textului este mai mica decat 1000;
- vectorul de cuvinte va avea maxim 100 de cuvine a 100 de caractere fiecare;
- delimitatorii pe care trebuie sa ii luati in calcul sunt: spatiu( ), virgula(,), punct(.), endline (\n)

- nu aveti voie sa folositi alta metoda de sortare in afara de qsort. In cazul in care veti folosi alta metoda punctajul pe acest task se va pierde;
- nu aveti voie sa folositi variabile globale, cu exceptia delimitatorilor de cuvinte. Pe acestia din urma ii puteti retine intr-o variabila globala.

Exemplu

```
number_of_words: 9
s: "Ana are 27 de mere, si 32 de pere."
dupa apelul get_words: words = ["Ana", "are", "27", "de", "mere", "si", "32", "de", "pere"]
dupa apelul sort: words = ["27", "32", "de", "de", "si", "Ana", "are", "mere", "pere"]
```

Pentru mai multe informatii despre qsort puteti accesa linkul: qsort [https://man7.org/linux/man-pages/man3/qsort.3.html]

Punctare

Task-ul are 25 de puncte, dintre care un punct pentru coding-style si detalierea implementarii.

Task-ul 4 - Conturi Bancare (25p)

Dupa a 4-a usa, Biju este intampinat de Thoralf Skolem, cel care a definit pentru prima data functiile recursive. Acesta ii da lui Biju contul bancar al lui Tzanca Uraganu, intr-o forma dubioasa, si-i cere sa scrie mai multe functii recursive, pentru a afla cati bani are Tzanca in cont.

Va trebui să implementaţi 3 funcţii mutual-recursive, care vor evalua o expresie primită ca parametru:

- `expression(char *p, int *i)` - evaluează expresii de tipul `term + term` sau `term - term`
- `term(char *p, int *i)` - evaluează expresii de tipul `factor * factor` sau `factor / factor`
- `factor(char *p, int *i)` - evaluează expresii de tipul `(expression)` sau `number`, unde number este o secvenţă de cifre

Precizări

- `p` este şirul de caractere
- `i` este poziţia actuală în şirul de caratere (atenţie, acesta va trebui actualizat în funcţii)
- numerele vor fi întregi pozitive, însă în urma operaţiilor pot apărea numere negative
- împărţirile vor fi făcute pe numere întregi
- rezultatele se încadrează pe tipul `int`
- expresiile primite vor fi corecte
- recursivitatea mutuală presupune apelarea reciproca a doua sau mai multe funcţii recursive. Ilustrativă în acest sens este următoarea secvenţă de cod:

```
function1()
{
    // do something
    function2();
    // do something
}
```

```
function2()
{
    // do something
    function1();
    // do something
}
```

- pentru mai multe detalii legate de recursivitatea mutuală consultaţi acest link [https://en.wikipedia.org/wiki/Mutual_recursion].

Orice rezolvare care nu foloseşte recursivitate mutuală nu va fi punctată

Exemplu de test

test.in	test.out
15+8*2000	16015

- Funcţiile pe care trebuie să le implementaţi se vor apela între ele
- Pentru împărţiri cu semn: cdq [https://stackoverflow.com/questions/36464879/when-and-why-do-we-sign-extend-and-use-cdq-with-mul-div]

Punctare

Task-ul are 25 de puncte, dintre care 2 puncte pentru coding-style si detalierea implementarii.

Task-uri bonus

Dupa a 5-a usa, Biju il gaseste pe Steve Jobs, iar in spatele lui vede si mai multe usi. Steve ii spune ca a ajuns in zona in care isi poate dovedi cu adevarat valoarea. Dupa fiecare usa va gasi un plic, in care va fi o provocare cu care nu s-a mai intalnit pana acum. Steve ii ofera mai multe chei, fiecare cheie putand deschide una din usile din spatele lui. Ii mai spune ca unele usi inca nu pot fi deschise, dar vor putea fi deschise curand.

Ca task-uri bonus, aveti de ales din urmatoarele:

- assembly pe 64 de biti (10p)
- instructiuni speciale - cpuid (10p)
- sintaxa AT&T (10p)
- instructiuni vectoriale (15p)

Vor aparea task-uri aditionale pe parcursul desfaurarii temei.

Cerintele task-urilor bonus si punctarea lor se pot gasi in fisierele task_*.md, din cadrul folderului fiecarui task.

Trimitere și notare

Temele vor trebui încărcate pe platforma vmchecker [https://vmchecker.cs.pub.ro/ui/#IOCLA] (în secțiunea IOCLA) și vor fi testate automat. Arhiva încărcată trebuie să fie o arhivă `.zip` care să conțina, **in radacina arhivei**:

- task1.asm
- task2.asm
- task3.asm
- task4.asm
- README (optional)
- sursele task-urilor bonus pe care le implementati

Notarea fiecarui task este detaliata in cadrul descrierii task-ului respectiv.

Pentru punctele de la "Descrierea implementarii si coding style", vi se cere sa descrieti implementarea voastra prin comentarii in cod si, eventual printr-un README. README-ul nu este obligatoriu pentru aceasta tema, ci se va pune accent pe aspectul general al codului. Cititi si sectiunea de Reguli si notare [https://ocw.cs.pub.ro/courses/iocla/reguli-notare], in special cea referitoare la "Reguli de realizare a temelor".

Punctajul maxim posibil este de 150, dintre care 50 de puncte bonus. Orice punctaj peste aceasta valoare va poate aduce glorie eterna, dar, in catalog, va fi trunchiata la 150.

Checker-ul poate fi rulat din folderul tema3, executand comanda `make checker`.

Mașina virtuală folosită pentru testarea temelor de casă pe vmchecker este descrisă în secțiunea Mașini virtuale din pagina de resurse.

Precizări suplimentare

Scheletul temei 3, checker-ul si cerintele fiecarui task pot fi gasite pe repository-ul [https://github.com/systems-cs-pub-ro/iocla] de IOCLA, in folder-ul `teme`.

Toate task-urile vor fi realizate in limbajul de asamblare x86 (32 de biti), in afara de cazurile cand se specifica explicit alt limbaj.

ACS meme lord:

Resurse

- Reguli si notare [https://ocw.cs.pub.ro/courses/iocla/reguli-notare]
- Github IOCLA [https://github.com/systems-cs-pub-ro/iocla]

iocla/teme/tema-3.txt · Last modified: 2022/01/09 20:36 by ilinca_ioana.strutu