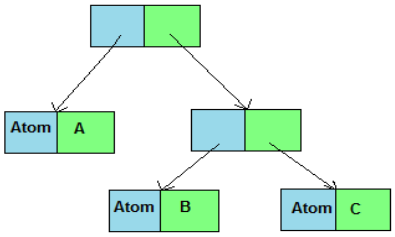
In the name of God

PL homework #4

PanteA Habibi - 9131010

3.1)

a)



b) (cons (cons ‘A (cons (‘B ‘C)) (cons (‘B ‘C))))

The reason that we have two cons cells with the same parts is that each evaluation of (cons ‘B ‘C) creates a new cell.

c) ((lambda (x) (cons ((cons (‘A x)) x)) (cons ‘B ‘C))

Here we first evaluate (cons ‘B ‘C) to produce the cons cell, then passing the cell to the function. That creates the upper cell with two pointers.

3.2)

a)

It's not possible, because there is no way to understand that an expression is undefined. We should continue executing, if an expression is undefined the system will halt and we won't get that this expression is undefined to skip.

b)

In order to do these, we should have a concurrent machine for running all expressions in parallel. Then each of them at first returns true, will determine the return value of cond and if none of them results in true the result will be undefined.

c)

```
(defun odd(x) cond((eq x 0) nill)
        ((eq x 1) t)
        ((> x 1) (odd(- x 2)))
        ((< x 0) (odd(+ x 2)))
        (odd( + x 1) nil))
)
```

d)

It's better to implement SCOR using (a), because in (a) expressions evaluated sequentially. But in (b) if there is one true statement the result will be true, and we need the result to be undefined.

POR is more easily implemented using (b) because in (b), expressions are evaluated in parallel just as we need in POR. But it is difficult using (a) because in (a) if there is one undefined statement the result will be undefined but we need the result to be true if even one of the expressions evaluate to true.

3.4)

a) f (h xs)

b) i: maplist

ii: car

c) compose(f, h)

3.5)

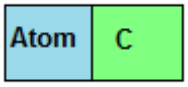
a) No, because it is possible that a list from a position to position can not be accessed by our program but other programs have access to it or the pointer is already exist but in several run time of the program , it can not have access to it. or there exist a program which can have access to it and with this algorithm it is not recognized as garbage or as an example “inline” functions after the definition and using them even though the pointer is already exist in memory, it is not possible to refer them and use them.

b) Yes, because each program can be continued by different arrangement of car and crd s. so if by none of the arrangement of the running can not be accessed do it can not be accessed by car and crd of base registers.

c) it is not possible to find out which of the memory cells has never been used in different running of the program simultaneously with different arrangement before the run time and without overhead.but with some approaches like creating memory graphs of inter-programs, we can write collectors and further more sometimes we should hold some states of the variables like counter of the references.

3.6)

a) All the cons cells are deleted, except the result:



b) (lamda(x) (cons (replaca x (‘a ‘b)) (car x)))) (cons (‘c ‘d) (‘e ‘f))

3.8)

a) $T(n) = 3 + \max (T(n-1), T(n-2)) = 3 + T(n-1), T \in O(n)$

b)

(cons (rplaca x ‘a) car x)

(cons (future(rplaca x ‘a)) car x)

In the second case if the “car x” is executed earlier than “rplaca x ‘a” then its value will differ with the value returned from the first one.

c) Consider an OR expression that it's implemented as "if an expression is true it's not important what will the second one be: the result will be true."

Now consider the expression “e_1 OR e_2” in which e_1 evaluates to unknown and e_2 evaluates to true. So the returning value of “e_1 OR e_2” will be unknown while the value of “future (e_1) OR e_2” will be true

d)

When order of the errors is important, we won't be able to set the priority of different errors and concurrency will cause some problems.

In every execution we may face a new error (i.e. if there are different errors in expression e each time the program is executed, a different handler may be executed because they are running in parallel and the one which ends first will be executed)

پیاده سازی select

```c
#include <stdio.h>
typedef struct cell cell;
struct cell{
    cell *cdr;
    enum { ATOM, NATOM } type;
    union {
            int atom_value ;
            cell* car_ptr;
    } car;
};
cell* select(cell*x ,cell*lst){
    cell* ptr;
    for(ptr=lst;ptr!=0;){
        if(ptr->car.atom_value== x)
            return ptr->cdr;
        else
            ptr=ptr->cdr;
    }
    return NULL;
}
cell* select1(cell*x ,cell*lst){
    cell* ptr , *previous;
    for(ptr=lst;ptr!=0;){
        if(ptr->car.atom_value == x)
            return ptr->cdr;
        else{
            previous=ptr;
            ptr=ptr->cdr;
            free(previous);
        }
    }
    return NULL;
}
```

```c
main(){
    cell* list= malloc(sizeof(cell));
    int d;
    printf("cell1:");
    scanf("%d",&d);
    list->car.atom_value=d;
    list->cdr=NULL;
    cell*head=list;
    int i=0;
    for( i=0;i<9;i++){
        printf("cell%d:",i+2);
        scanf("%d",&d);
        cell* new_cell= malloc(sizeof(cell));
        new_cell->car.atom_value=d;
        new_cell->cdr=NULL;
        head->cdr=new_cell;
        head=new_cell;
    }
    head=list;
    printf("\nThe list :[");
    for( i=0;i<10;i++){
        printf("%d ",head->car.atom_value );
        head=head->cdr;
    }
    printf("]\nEnter x:");
    scanf("%d",&d);
    head=select(d,list);
    printf("Result:[");
    if(head==NULL)
        printf("not in the list or the last element.");
    while(head!=NULL){
        printf("%d ",head->car.atom_value );
        head=head->cdr;
    }
    printf("]\n");
    head=select1(d,list);
    printf("Result:[");
    if(head==NULL)
        printf("not in the list or the last element.");
    while(head!=NULL){
        printf("%d ",head->car );
        head=head->cdr;
    }
    printf("]\n");
}
```

گزارش garbage collection

نوعی مدیریت حافظه ی خودکار است. این روش حالت خاصی از مدیریت منابع است که منابع محدود حافظه در آن که مدیریت میشوند و تلاشی برای بازستانی و بازیابی زباله یا حافظه ای که توسط اشیاء به کار گرفته شده و دیگر مورد نیاز برنامه نیست، خواهد بود . تکنیک زباله روبی توسط جان مککارتی برای حل مشکلات لیسپ اختراع شده است.

زباله روبی اغلب در مقابل تنظیم دستی حافظه قرار دارد که با استفاده از آن ، برنامه نویس باید مشخص کند چه زمانی کدام قسمت های حافظه باید مورد استفاده قرار گیرند و کدام قسمت ها باز پس داده شوند و به حافظه اصلی برگردانند. اگر پروسه ای فضای مورد استفاده اش را رها سازی نکند ، برنامه تا انجا ادامه می یابد که یا کار پروسه به پایان برسد و یا کل حافظه اشغال شود و امکان فراهم سازی بقیه ی حافظه ی مورد استفاده ی برنامه وجود نداشته باشد . از طرفی اگر این کار به صورت دستی انجام پذیرد و به عهده ی برنامه نویس گذاشته شود ، ممکن است مشکلاتی از قبیل آزاد سازی زودتر از موعد حافظه، درخواست حافظه کمتر از میزان مورد احتیاج و یا عدم رهاسازی حافظه در زمان لزوم و ... رخ دهد . بررسی ها نشان می دهدکه فرایند gc می تواند زمان اجرای برنامه را تا ۱۱ درصد زیاد کند .

فرایند gc را می توان به دو بخش اصلی زیر تقسیم نمود:

1- تشخیص حافظه های زباله و تمییز آنها از حافظه ی مورد استفاده ی برنامه

2- بازگرداندن حافظه به حافظه ی آزاد قابل دسترسی

برای این منظور روش های مختلفی مورد استفاده قرار می گیرد که در ادامه معرفی می شوند:

Reference counting

هر object ، با یک شمارنده برای اشاره گر ها همراه می شود.

هر بار که پوینتری به آن شی ساخته می شود، یکی به آن شمارنده اضافه شده و اگر پوینتری حذف شود ، شمارنده یک واحد کم می شود.

اگر شمارنده به صفر برسد، حافظه ی مربو به آن شئ زباله در نظر گرفته شده و می تواند رها سازی شود .

هرگاه حافظه ی مربوط به یک شئ بازیابی شود، تمام فیلد های آن بررسی شده و شمارنده ی تمامی اشیایی که به آنها اشاره میکرده ، یک واحد کم می شوند.

این روش دو مشکل اصلی دارد :

نمی تواند در ساختار های حلقه ای به درستی عمل کند.

سربار زیادی برای نتظیم شمارنده ها بر سیستم پیاده می شود.

در مجموع می توان گفت این روش برای سیستم های high-performance مفید نیست ولی در بسیاری از سیستم ها که از ساختار های acyclic و بدون حلقه استفاده می کنند به کار گرفته می شود زیرا پیاده سازی ساده و قابل فهمی دارد .

Mark-sweep collection

با شروع از ریشه و پیمایش گراف وابستگی اشاره گرها حافظه هایی که پیمایش می شوند عالتم زده می شوند تگ 1 و پس ازاتمام پیمایش تمام مکان هایی از حافظه که عالمت زده نشده باشند ، رهاسازی می شوند .

این الگوریتم نیز مشکالتی دارد :

ایجاد fragmentation در حافظه که باعث می شود تعریف شئ جدید حجیم سخت شود.

هزینه آن وابسته به اندازه Heap است . فرایند sweep به صورت تنبل انجام می شود و حافظه ی رها شده جمع آوری نمی شود .

Mark-compact collection

در این روش حافظه های خالی در انتهای مموری جمع آوری می شوند .

این روش نیازمند گذرهای بسیاری بر حافظه است. یکی برای تشخیص مکان جدید اشیا و به تبع آن چندین گذر برای تغییر اشاره گر ها و جابه جا کردن شئ . این الگوریتم در مقایسه با روش قبل می تواند بسیار آهسته تر باشد .

این جمع آوری به رو روش two finger برای اشیای هم اندازه با دو گذر و الگوریتم لیسپ 2 برای متغیرهای غیرهم اندازه وبا استفاده از یک اشاره گر اصافه برای هر شئ ، انجام می گیرد.

Copying garbage collection

همانند الگوریتم mark-compact ، این الگوریتم تمام اشیای زنده را به یک نقطه از حافظه برده و در نتیجه باقی حافظه آزاد است و می تواند مورد استفاده قرار گیرد.

در این روش حافظه به دو semispace تقسیم شده که در هر لحظه فقط یکی مورد استفاده است. این روش می تواند به صورت stop and copy انجام پذیرد که به صورت زیر عمل می کند:

حافظه به صورت خطی از قسمت مورد استفاده ی فعلی به برنامه ی درخواست کننده اختصاص داده می شود . اگر حافظه مورد درخواست بیش از فضای آزاد قسمت فعلی باشد، برنامه متوقف شده، gcصدا زده می شود .

Non-copying implicit collection

این روش به دو فیلد اشاره گر و یک فیلد رنگ اضافه برای هر شئ نیاز دارد .

اشاره گر ها برای اتصال هر قسمت از حافظه به یک لینک لیست دو طرفه به کار می روند.

فیلد رنگ به آن منظور استفاده می شود که مشخص کند هر object متعلق به کدام دسته داده است .

امکان compact کردن حافظه را ندارد . بر خالف روش قبل برای trace کردن حافظه ی زنده و مورد استفاده نیاز به دو فضای to و from نیست . به همین منظورهزینه ی حافظه ی کمتری نسبت به روش قبل دارد .