

Lecture 06

Lecture contents:

- 1-Revision on past lecture
- 2-Disadvantages of static lists
- 3-memory sections (program code – heap – stack)
- 4-Pointers
- 5-Pointers & arrays
- 6-Pointers & structures

بدأنا نتكلم عن ال lists المحاضرة اللي فانت وقولنا انها عبارة عن array وبحاول اتغلب على مشاكلها عن طريق اني ماخليش ال user يقدر بـ acces لها غير عن طريق insert – delete – retrieve functions بتعمل ولقينا ان احنا محتاجين نعرف size ال array دايماً فعملنا اسمه variable .. وعشان نعمل insert يكون فيه حاجة تقولنا ال array اللي جوة دي فيها كام دلوقتي عشان نعرف هنقدر ن insert ولا لو هنقدر بيقى هن insert فين في ال array بالظبط .. وال حاجة دي سميئناها NumberofItems بعد كدة قولنا عايزين نعمل retrieve first element ولقينا اننا مش محتاجين variable او مؤشر جديد بعد كدة قولنا طب نعمل retrieve next element ولقينا اننا محتاجين مؤشر جديد يقولنا فين ال next element دة .. وسميئناه currentPosition

بعد كدة كتبنا ال return وال pre & post conditions على اساسهم كتبنا الكود :

first

Pre-conditions:

List can't be empty (numberofItems > 0)

Post-conditions:

Current position has a value = 0

Item is retrieved to a variable

Return:

True if retrieve succeed (list contains at least 1 item)

False if failed

```
bool List::first (listElementType &e){  
    if (numberofItems <= 0) {  
        return false;  
    }  
    else {  
        currentPosition = 0;  
        e = listArray[currentPosition];  
        return true;  
    }  
}
```

الأسماء اللي في الكود اللي فوق دي الهدف منها اننا نبقى عارفين ان ال pre & post conditions دول مش كلام بنكتبه عشان ال documentation وخلاص .. لأننا لازم اتحقق في الكود وتأكد انه مطابق مع ال conditions بتعاتي

احنا عرفنا الفكرة فهنعمل بسرعة :

next

Pre-conditions:

First has been executed at least once (currentPosition ≥ 0)

There is 'next' item in the list (currentPosition $\leq \text{numberofItems} - 1$)

Post-conditions:

currentPosition is incremented

e gets the value stored in listArray [currentPosition]

Return:

True if 'next' succeeds

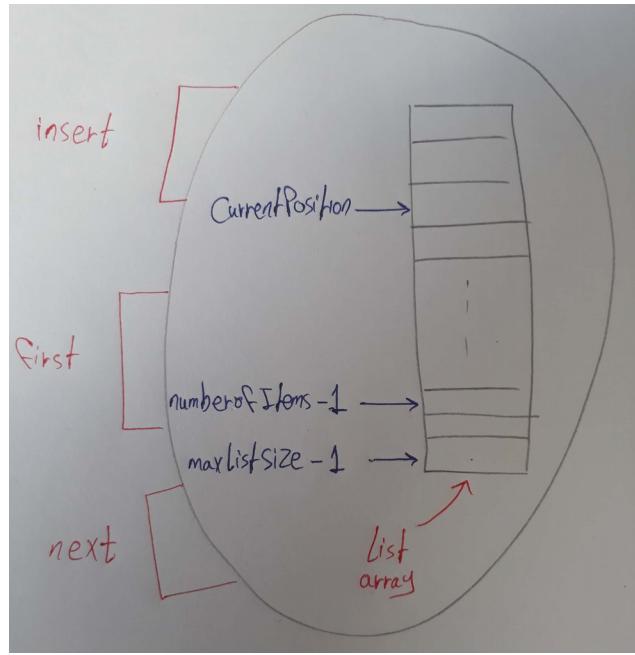
false otherwise

```
bool list::next (listElemType &e) {  
    if (currentPosition < 0 || currentPosition == numberofItems-1){  
        return false;  
    }  
    else {  
        currentPosition++;  
        e=listArray[currentPosition];  
        return true;  
    }  
}
```

خلي بالك احنا بنـد check على عدد ال array مش ال list elements يعني بنـد check على elements numberofItems مش maxListSize

وخلـي بالـك في ال next اـحـنا بنـشـاـور عـلـى الـحـاجـة الـجـديـدة وـبـعـدـين بنـقـرـاـها .. بـسـ في ال insert بنـد insert الاول وـبـعـدـ كـدـةـ نـشـاـور عـلـى المـكـانـ الجـديـدـ

من الاخر بـقـى اـحـنا كـدـةـ عملـنـا ال black box بتـاعـنـا وـنـجـحـنـا انـنـا نـخـلـي ال user ماـيـعـلـشـ access لـل array غير بشـروـطـ وـطـرقـ معـيـنةـ :



اللي احنا عملناه دة كان اسمه static linear list .. فيه بقى حاجة تانية الدكتور اتكلم عنها المحاضرة اللي فانت اسمها circular list ارجع شوفها

بس بالنسبة لل static list .. احنا عندنا ليها 3 عيوب رخمين جداً ..

Disadvantages of static lists:

1-Fixed size

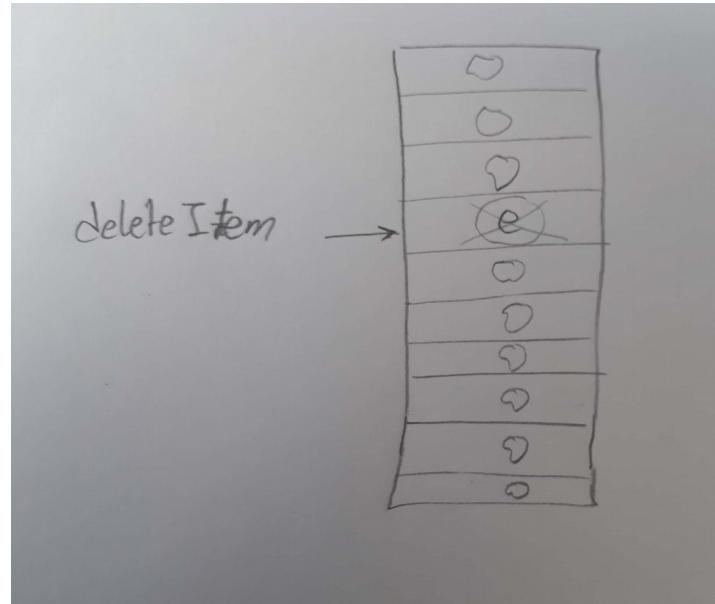
يعني انا مثلًا عملت array .. ال size دي لازم بيديها ال size بتاعها، نقول مثلًا هنخلي ال size بتاعها 1000 ... وال 1000 دول انا خلاص مدام عملتهم ببقى مقدرش اشيلهم او ازودهم او اقلهم طب افرض انا عندي application محتاج ال 1000 دول بيقوا 10000 مثلًا .. مش هقدر اعمل كدة غير لما اقفل ال application وافتتح الكود واخلي ال 1000 دللي 10000 واعمل compile وبعدين اشغل ال application تاني .. او اتصرف واعامل بال 1000 دول وخلاص وساعتها طبعاً ال client program مش هي ... run efficiently ... ممكن اقول طب خلاص انا هعمل size ببقى generic لأي حد، هخليه 100000 مثلًا وأكيد مفيش حد هيحتاج size اكتر من كدة .. بس دة حل غلط جداً طبعاً لأنه هياخذ من الميموري مكان كبير او ي وال user غالباً مش هيحتاج ال size دة اصلاً .. ودة اساساً لو ال size دة عدى من ال compiler ما يعمل ايرور، فالموضوع دة غلط خلاص وافرض برضه ان انا مش محتاج ال 1000 دول كلهم .. محتاج 500 بس، فانا ليه هجز 1000 مكان عشان انا محتاج 500 بس منهم؟؟؟

فانا عايز احل المشكلة دي عن طريق اني اخلي ال user يقدر يعمل ال size بتاعه بسهولة في ال runtime

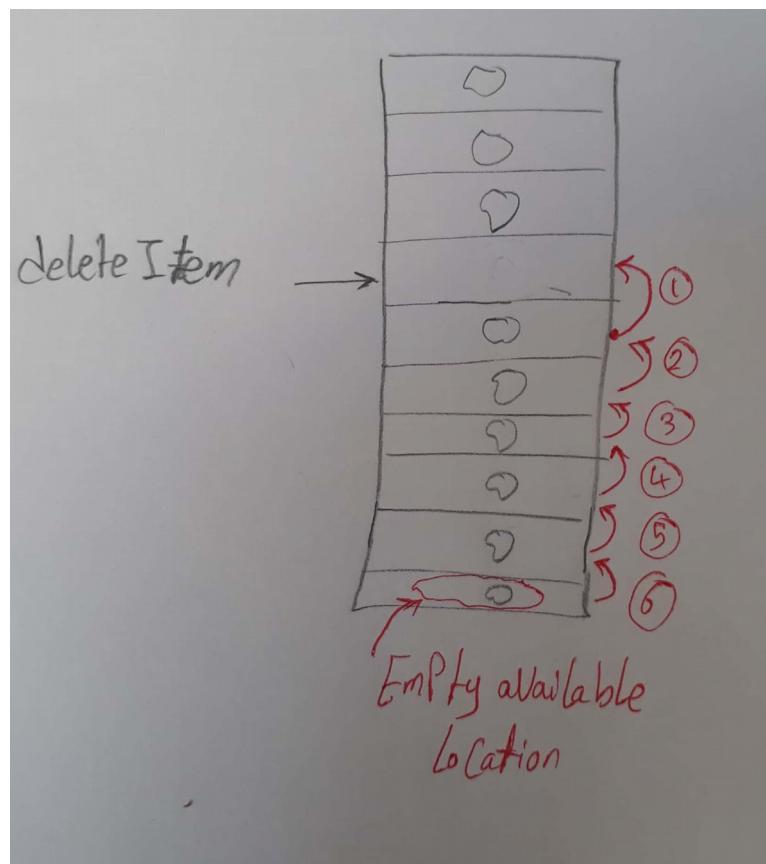
2- Delete is very inefficient .. “we've to move large number of items”

الدكتور قال نعمل احنا ال delete في البيت بس قال فكرتها .. لو انا عايز اعمل delete لحرف ال e في ال list delete (e)

ساعتها هيكون فيه مؤشر بيبداً من اول element ويدور على حرف ال e لحد لما يلاقيه ويمسمه .. زي كدة



فكرة بقى فيه element فاضي في النص ودة ماينفعش لأننا عشان نعمل insert تاني في المكان الفاضي لازم يكون المكان الفاضي دة هو اخر مكان في ال list يعني ماينفعش ن insert في مكان في النص كدة .. فاييه الحل؟؟
 الحل اننا نخلي المكان الفاضي اللي موجود بيقى في الآخر بدل ما هو في النص .. ازاي؟؟ هنعمل shift لكل ال elements اللي تحت المكان اللي اتمسح .. كل element يطلع على ال element اللي فوقه عشان في الآخر بيقى المكان الفاضي هو اخر element في ال array .. زي كدة (ببدأ ن shit من رقم 1 لحد ما نوصل لرقم 6) :

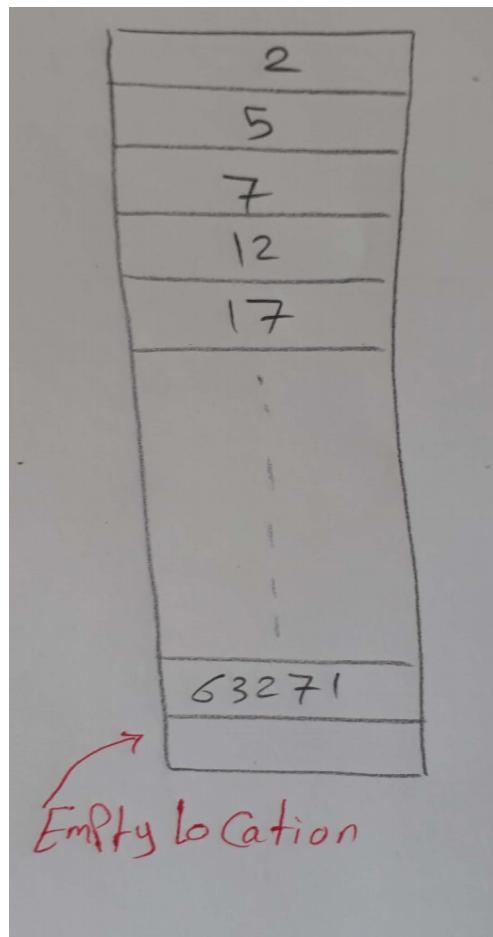


طبعاً حاجة زي دي مش هنحس انها رخمة لو كنا شغالين ب arrays صغيرة .. بس تخيل مثلاً لو عندنا array فيها 1000000 elements وانا عملت لل delete element رقم 100 .. وبالتالي انا محتاج انقل 999900 element وطبعاً دة استهلاك لل execution time ولليلة كبيرة .. فالموضوع مش efficient على الفاضي و resources

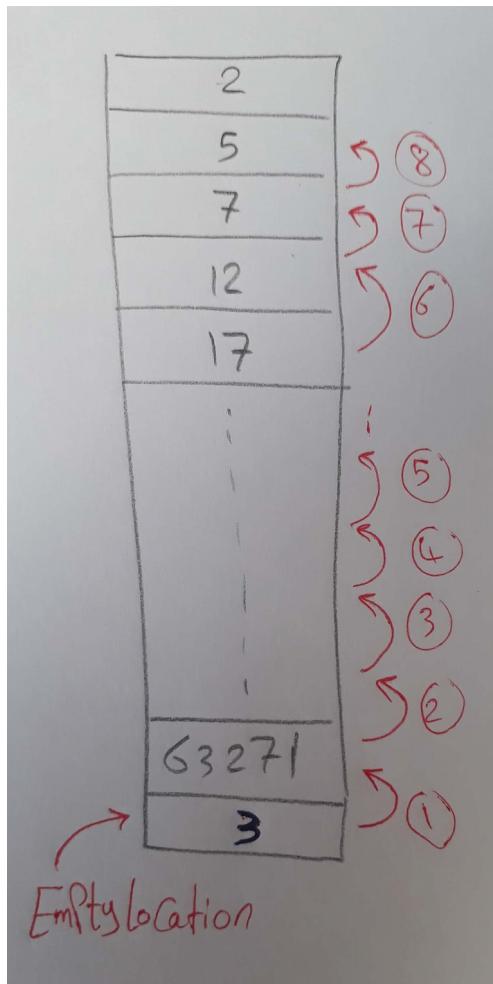
3-Insertion into ordered (sorted) list is very inefficient

الدكتور قال النقطة دي مهمة وبيجيها في الامتحانات

هنا بنقول ان احنا مرتبين ال list بتاعتنا بترتيب معين .. قول مثلاً ترتيب تصاعدي في المثال بتاعنا زي كدة:



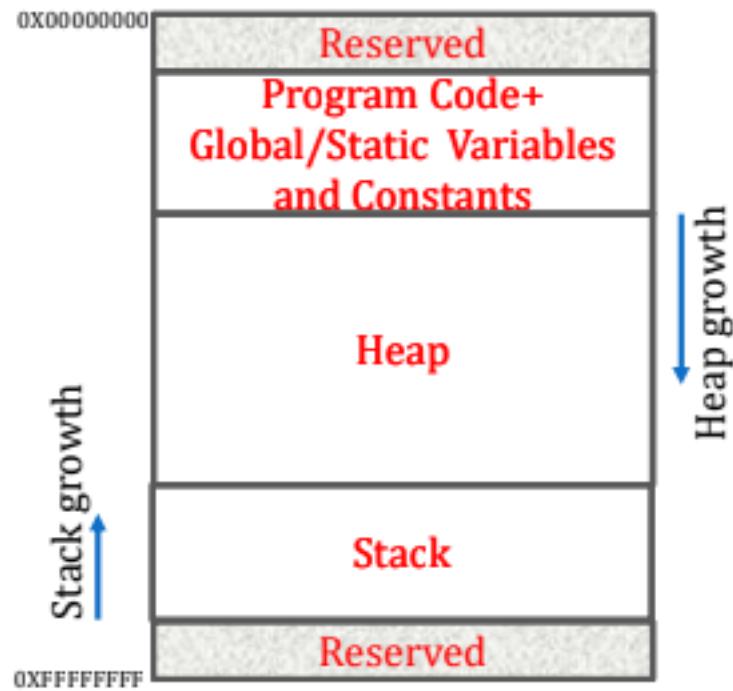
وجينا قولنا عايزيين نـ insert رقم 3 في ال list .. كتبنا كدة مثلاً ((insert (3)) .. فأكيد محتاجين نـ insert الرقم دة في مكانه الصح (مش هـ insert في اخر مكان وخلاص زي ما حانا متعودين)
فاحنا كدة يا معلم محتاجين في الاول نفصل مكان بين ال 2 وال 5 عشان نعرف نحط فيه ال 3 ، بس مش هنقدر نعمل كدة .. فخلاص نحط ال 3 في المكان الفاضي تحت خالص ونبـأ نشفتها واحدة واحدة من تحت ل فوق لحد ما نخليها بين ال 2 وال 5 :



وطبعاً دة برضه مش efficient عشان كدة عامل مشكلة
 فكدة مشاكل ال list هي نفس مشاكل ال array اللي كانت عندنا زمان .. فاحنا دلوقتي عايزة نحل مشاكل ال array دي، عشان كدة
 هنأخذ ال dynamic allocation
 بس في الاول محتاجين نراجع على memory maps كدة زي ال concepts وال : memory maps

Memory sections:

دلوقتي هن بص على شكل ال memory والتقسية بتاعتها عشان الكود اللي بنكتبه يستغل
 ال memory map بتبقى منقسمة لـ section :
 Program code – Heap – Stack – reserved sections - ...



الـ reserved sections دي تكون system dependent سواء OS او سیستم عادی، بس غالباً اول کام address و اخر کام address ببیقوا reserved addresses مایفععش نلعب فيهم .. طب لو لعینا فيهم؟؟؟
الـ compiler مش هيقول حاجة لأنه شایف انه valid addresses عادي .. بس طبعاً دة غلط، طب ايه بقى اللي هيحصل؟؟؟ الاجابة
ان دي حاجة system dependent الله أعلم بيهها بس خلیک متأكد انك لو لعنت في الـ reserved sections بیقی انت کدة بتعمل مصيبة سودا

تمام كدة .. ندخل على ال program code السكشن دة من اسمه كدة بيتحزن فيه ال program code وده بيقى Read only عشان مايحصلش اي تغيير بالغلط في ال runtime وال size بتاعه بيقى fixed

فيه بقى data section : دة بيتخزن فيه ال static local variables وال global variables .. لو كانواInitialized
هيتخطوا في سكشن اسمه data.
ولو كانوا uninitialized هيتخطوا في سكشن اسمه bss ... الدكتور ماتكلمش عن ال data وال bss وقال لو حد عايز يعرف
عنهم اكتر يسبرش

و فيه ال stack : دة ببقي flexible size R/W يعني ببقي على حسب ال program .. غالبا هو بببدأ من ال address الكبير ويفضل ي decrementation call activation for function calls وال function parameters وال local variables ال stack بيخزن ال return address (او بمعنى اصح ال address)

ولازم ال stack يفضل في بالي ك programmer دايماً عشان زي ما قولنا هو flexible size فممكنا لو ما هتميتش بيها يحصل ان ال stack يدخل في ال heap او في ال data section ويحصل segmentation fault

آخر سکشن هنترکم عليه بقى واللى اتكلمنا عن القصة اللي فوق دى كلها عشانه هو ال Heap : دة برضه size R/W و flexible size ، stack .. بس استخدامه الوحيد بيقى في ال dynamic memory allocation .. وكمان غالباً ال heap بيتملى عكس ال stack

يعني لو ال stack بتاعنا بيتملي من تحت لفوق (ال address decremented) يعني ال heap هيتتملي من فوق تحت (increment)
فيه بقى operations 2 في ال C++ بنسخدمهم لكل ال data types عشان نشتغل على ال Heap :

1- new

2- delete

وخليلك عارف اننا لو عرفنا نستخدم ال heap صح هيبي لذيذ جدا ويساعدنا كثير بس لو ما عرفناش نستخدمه هيبيوظنا البرنامج كله يا معلم

كدة خلصنا الكلام النظري .. تعالى بقى ناخد مثل على كود ونشوف ايه اللي بيحصل فيه :

```
main() {
    int i, j;
    f1();
    .... Rest of program
}

f1(){
    int i, k;
    f2(i,j);
}

f2(int u, v){
    int m, n;
    n = f3(m);
}

int f3(int i)
{
    int k = i*i;
    return (k);
}
```

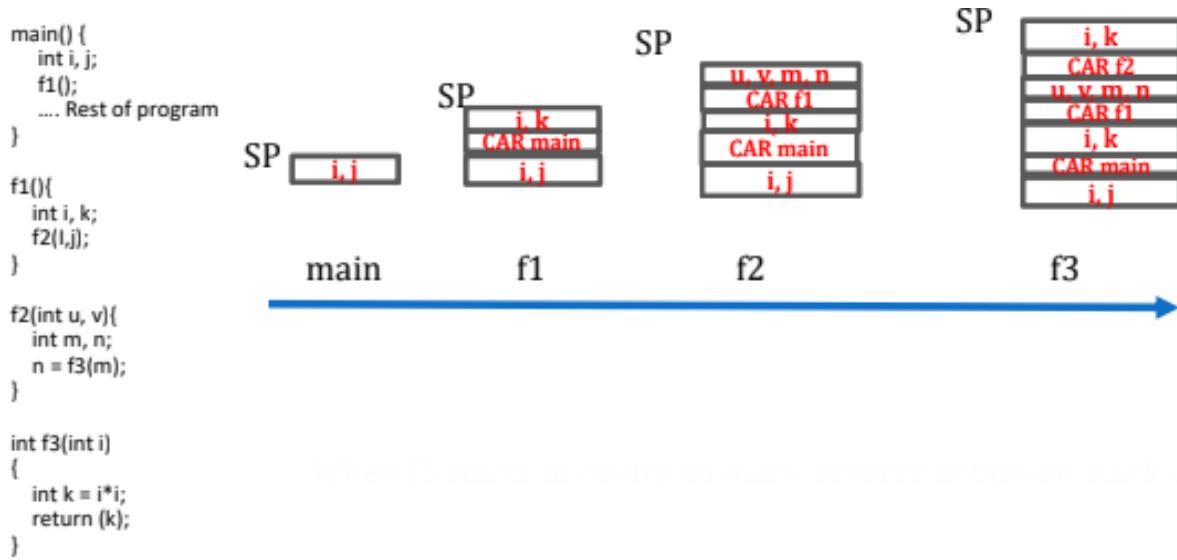
دة كود بسيط هنمشي وراه واحدة ونشوف ايه اللي بيحصل في الميموري مع كل خطوة بنعملها
في الاول الاستاذ compiler دخل على ال main .. شاف ان الاستاذ user عمل declaration 2 integer variables اسمهم i,j .
فيهixinz 1 في اول مكان فاضي في ال stack وبdecrement ال SP وبيخزن ال زوب decrement ال stack هيخزن I في
مكان و ز في مكان تاني يعني مش بعض زي الصورة اللي في ال slide (

بعد كدة هيلالي ان فيه function call اتعمل .. فيبروح ل f1 .. بس قبل ما يروح هو يحتاج بـ save ال address اللي هيرجعه
تاني لما يخلص ال function عشان يكمل باقي البرنامج .. ف هيـ save ال return address وبـ decrement save ال address
كدة ال compiler خلاص دخل f1 .. هيلالي ان فيه k & I اتعرفوا جيد فيبروح يخزنهم
بس ثواني .. كدة ال memory بقى فيها 2 variables ليهم نفس الاسم .. مش دة compilation error مش دة الا جابة لأ عشان كل
variable منهم له scope بعيد عن التاني يعني f1 مش شايفه I بنتاعت ال main .. وال main مش شايفه I بنتاعت f1 .. دلول 2 variables

المهم بعد كدة هيحصل call ل f2 فنهخزن ال f2 return address اللي جوة f1 عشان نعرف نرجعله تاني ..
لقينا ان f2 بتأخذ 2 arguments اسمهم u & v (ودول هيجوا من ز&I اللي كانوا في f1) .. فهـ save ال 2 دلول ..
دخلنا ال function ولقينا جوة فيه m &n فهـ save لهم برضه

لقينا f3 فهـ save ال f2 return address بتاع f2 .. و f3 بتأخذ argument اسمه I و I دة اللي هو m اللي في f2 فهـ save جديد

بعد كدة دخلنا جوة f3 ولقينا variable k جديد اسمه k .. فعملنا save لـ k بعد كدة لقينا return الحمد لله .. والصورة دي فيها توضيح التعلويذ اللي فوق :



بس ال compiler عايز يخرج بقى من الحاجات اللي دخل فيها دي عشان يكمل الكود بتاعه .. فهيرجع بالعكس لحد ما يوصل لل main .. يعني بدل ما كان بيزود الداتا اللي بتجيله من تحت لفوق .. لأ هيرجع يمشي تاني من فوق لتحت

فلما شاف return بتاعت f3 راح عمل i&k و بعد كدة راح لـ return address اللي كان مسيفه لـ f2 .. وهكذا هيفضل يمشي من فوق لتحت لحد لما يرجع تاني ل코드 الـ main ويكمـل الـ rest of program

طب احنا ليه بقى بنشرح كل الكلام دة؟؟ .. كل دة عشان ندخل على الـ dynamic allocation لأن الـ main focus لأن الـ dynamic instructions هيكون data structure في كورس الـ

نأخذ كود تانى نراجع منه على الـ stack pointers والـ

```

main() {
    int i = 5;
    int *pi; //defines pointer to int

    pi = new int; //Ask the heap library to reserve space for an int (4 bytes)
    assert(pi != NULL); //don't assume it will work. Null is illegal value for
pointer as it is in reserved section
    *pi = 2341; //assign the value 2345 to the location pointer to by
// Alternatively if (pi != NULL) *pi = 2341; else { handle the error}

    for (*pi=0; *pi < 10; (*pi)++)
        cout << *pi;

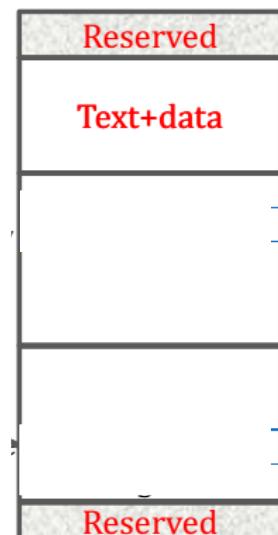
    delete pi; //Return location allocated to pi back to heap

    * pi = 8; //DISASTER: Don't ever use after de-allocation.
    // Will still compile

    // Only delete when no longer needed
}

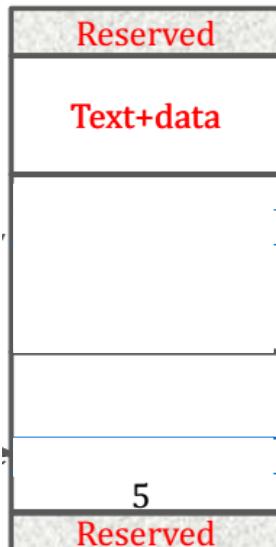
```

0x00000000



int i=5;

هخط الـ فى الـ .stack

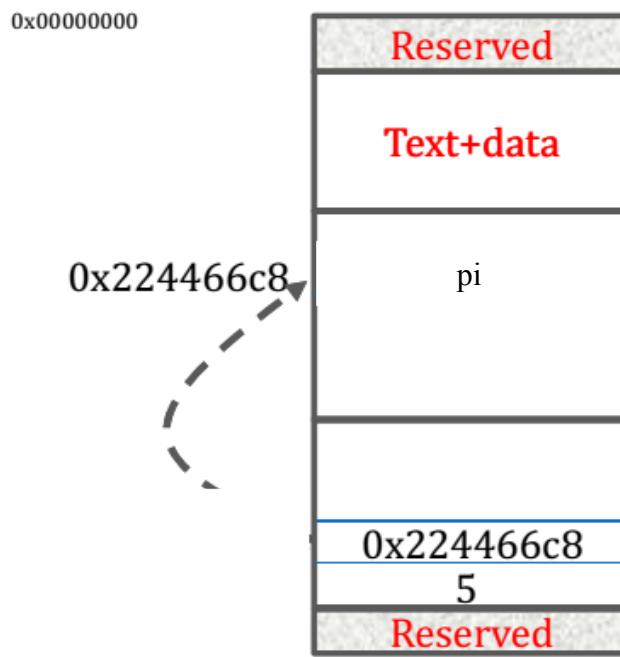


int *pi;

كده عملنا pi بি�شاور على حاجات int اسمه

pi=new int;

الـ new operator يشوف الـ data type حجمها د ايه ويروح للـ heap manager يقوله على الـ size اللي محتاجه.
فهنا مثلاً ليكن لقى address 0x224466c8 بالحجم اللي طلبه للـ pi والـ stack بناع الـ pi ده عنده.



assert(pi !=NULL)

دی بتتأكد اذا كانت ال new نجحت وللا لا ..

وهى ايه اللي يخلوها متتجحس !!؟

هنا مفيش حاجة تخليلها متتجحس عشان ال heap لسة فاضية خالص .. لكن فى برمج كبيرة ممكن تقفل ..

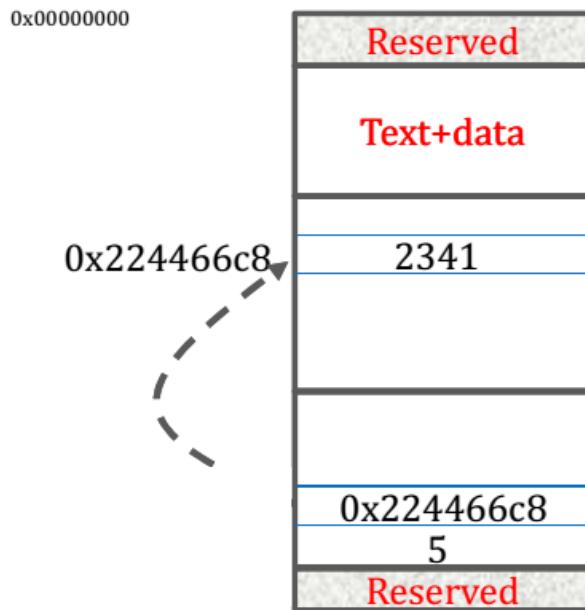
ف rule1

الدكتور قال لازم نحطها عشان فى البرامح الكبيرة ممكن تحصل كارثة .. فنتعود نحطها لما نيجى نكلم ال heap حتى لو البرنامج صغير.

*pi=2341;

لما احط * قبل pointer فى حته غير ال declaration بتاعه يعني حط فى ال address اللي بيشاور عليه ال pointer الحاجة دى.

فهنا هروح لل stack واشوف ال pointer بيشاور على address كام واروح ال address ده واحط فيه ال 2341.



ويمكن استخدم pointer بالـ* زى اى variable عادى.. واعمل for loop مثلًا و هكذا.
زى هنا

```
for(*pi=0; *pi<10; (*pi)++)
    cout<<*pi;
```

خد بالك

(*pi)++ is not the same as *(pi++)

اللى على الشمال دى اللي احنا مستخدمنها فى الـloop ودى معناها تزود الـvariable اللي جوة الـaddress اللي بيشاور عليه الـpi.

لكن اللي على اليمين يعني زود الـaddress اللي بيشاور عليه الـpi و هات الـpi اللي فى الـaddress الجديد ده.

واما نتيجى تزود الـaddress اللي بيشاور عليه الـpi بتشوف تعريف الـpi فهنا مثلاً كان int يبقى هزود 4 bytes مش واحد عشان ده اللي هو بيذخن 32bit فبيأخذ 4 .

تاني rule
لكل new فى الكود delete

delete pi;

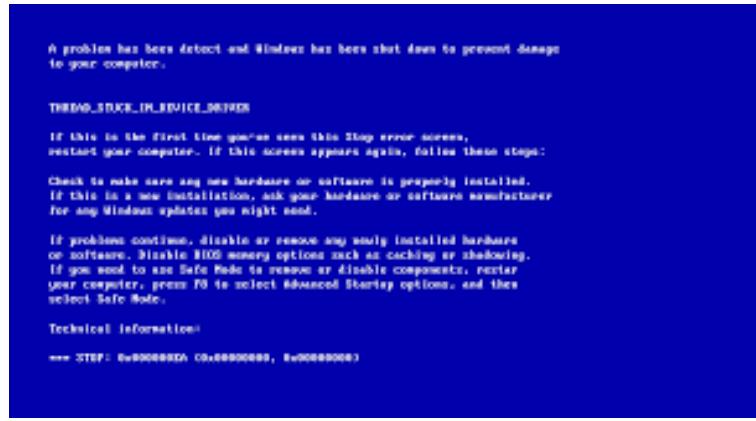
طب لو كتبت دلوقتى بقى

*pi=8;

مش هيرضي عشان احنا خلاص خلصنا و عملنا delete .. فمینفعش ن access variable بعد ما عملناه delete .

فخد بالك .. متعملش delete الا لما تتأكد انك مش عايز الـvariable اللي تاني .

كده الكود خلاص .. بس افرض فى كود تاني الـheap فضل يكبر لفتح و الـstack يكبر لفوق .. ايه اللي هيحصل؟ .. هتطلع الـblue screen of death اللي هى دى:



الحل ان يبقى فى linker يضبط بينهم.
وعشان كده لازم ببردو نحط delete لكل .new

فى بقى ميزة حلوة اوى لل pointers وهو انك ممكن تكتب address بعينه و تحط فيه variable بس لازم تكون careful جدا عشان غلط يخليك تكتب فى ال code segment او الكود بيتو اوتكتب مكان حاجة بتستخدمها و تبقى بوطت الدنيا.
زى هنا

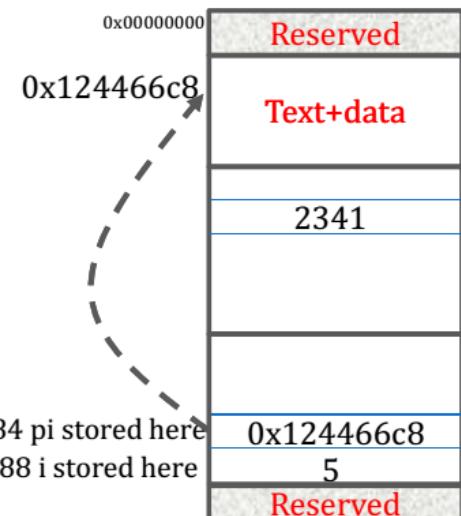
```

main() {
    int i = 5;
    int *pi; //defines pointer to int

    pi = 0x124466c8; // correct syntax but where is that address

    *pi = 5678; // Expect blue screen of death or segmentation
    fault. Don't do that
}

```



```

main() {
    int i = 5;
    int *pi; //defines pointer to int

    pi = &i; // let pi point to where i is stored

    *pi = 67;

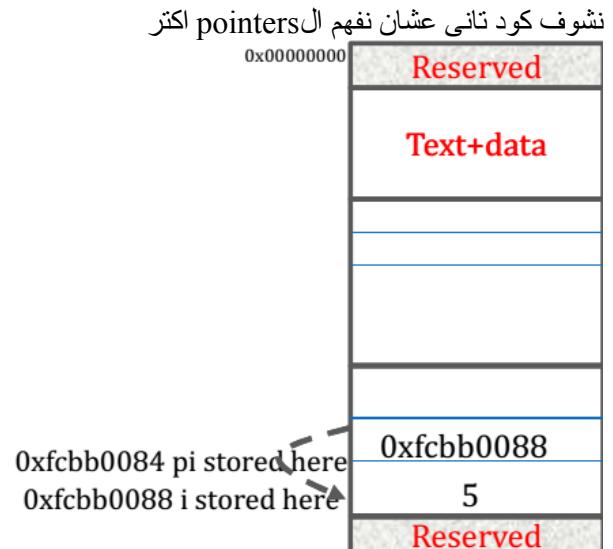
    cout << i ; // what will be printed?

    i= 15;
    cout << *pi; // what will be printed?

    //Now delete pi it is not needed

    delete pi; // Unknown effect
    //→ Blue screen of death or segementation fault
    // Delete only to be used when pointer allocated with new
}

```



pi= &i

1. &:address of

2. bit wise and

0x0001 & 0x0010

3. logic and

اللى كل bit يتعملها and مع ال bit اللي قدامها فهنا مثلاً تبقى النتيجة 0

اللى هى بتطابع true or false

نرجع للكود . كده ال pi بقت بتتشارر على المكان اللي فيه . فال stack بتابع ال ذا اللي هو 0xfcbb0088 address .

كده ال 5 هنتشال ويتحط مكانها 67

cout<<*pi;
the output:

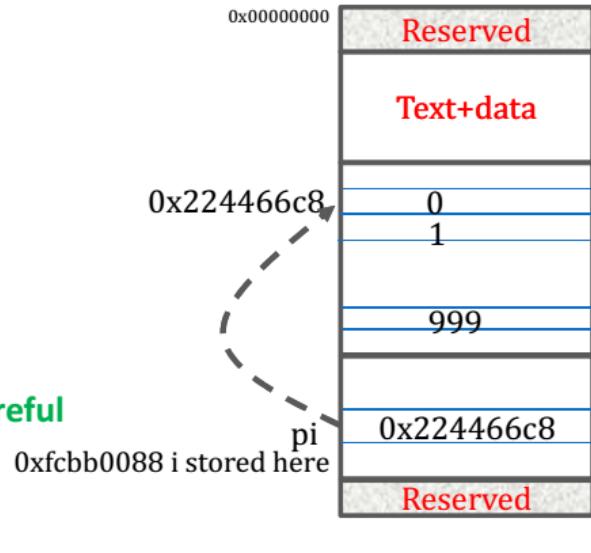
```
i=15;
cout<<*pi;
the output:
15
```

```
delete pi;
```

ايه ده هو احنا عملنا new فى اول الكود اصلا؟؟؟ لا .. يبقى هيدينى ال blue screen of death فدى غلط

نأخذ مثل تالت arrays مع ال pointers

```
main() {
    int i ;
    int *pi; //defines pointer to int
    pi = new int [1000]; // allocate array of 1000 ints to pi
    assert (pi !=NULL);
    for (i=0; i < 1000; i++)
        pi[i] = i; // pi can be used as array
    //Alternatively
    for(i=0; i < 1000; i++)
        *(pi+i) = i * i;
    //OR
    for(i=0; i<1000; i++)
        *(pi++) = i*i; //but here pointer moves. Be careful
    delete [] pi; //de-allocate all elements
    // delete pi wil deallocate first one only
}
```



الجديد هنا بردو

شافت new دى .. يعني اطلب من ال heap يجيب مكان فى ال memory بتاعته بس continuous array لازم بقى memory بقى ال بتاعته بس rule 1 لازم assert بعدها

وكل ما كنت طماع وكاتب رقم اكير لل array كل ما كان لازم اكتر تعمل assert .

ايده !؟؟؟؟ ولا حاجة الحقيقة ان ال array ما هو الا pointer abstraction حتى لما عملنا المرة اللي فانت list كده

ListArray[MaxListSize];
.pointer ده اصلا وعشان كده تانى for loop مكتوبة فى السلايدز هى الأصل

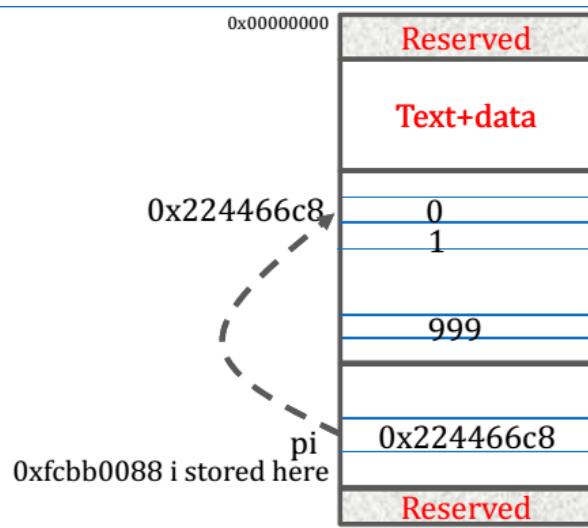
طب وتالت طريقة؟؟؟ لا تالت طريقة غلط عشان اول مرة هيدخل ال loop pi وبعدها يحط القيمة ف kedde سينا اول مكان فى ال array فاضى. وطبعا مننساش نعمل delete فى ال آخر

طيب هو انا ينفع اعمل ال loop على 2000 مثلا وانا معرفة ال array فوق ان ليها 1000 مكان بس؟؟؟

يتفق .. لكن logic syntax مينفعش عشان المكان الزيادة اللي عايز تاخده ده مش بتاعك و متحجزش لـ array من الاول .
 طيب بمناسبة الـ delete اللي بنأك عليها بعد كل new .. هو ممكن هنا مثلاً ابقى مش هستخدم اول 100 مكان فى الـ array بس الباقي
 لستة عايزه فامسح اول 100 مكان بس..؟؟؟

اه

```
delete [100] pi; //de-allocate first 100 elements
pi = pi+100; //now point to 1st element in rest of 900 elements!
```



كده المحاضرة خلصت بس الدكتور قال نبقي نি�ص على الـ pointers & structures اللي هما اخر two slides .. اهم بالمرة

Pointers and Structures/Classes

11

```
main() {
    Complex x;
    Complex *cp; //defines pointer to Complex class

    cp = new Complex(10.6,4.3); // allocate a complex ADT calling constructor 2
    assert (cp !=NULL);
    cp->ReadComplex(); //Call member function using arrow -> operator
    x.ReadComplex();
    cp->Add(x);
    //OR
    (*cp).Add(x);
    delete cp;

    cp = &x; // possible cp still alive
    cp -> Add(x); //No problem
}
```

pi = new Complex[2000];



Pointers and Structures/Classes

12

```
main() {
    struct Test{
        int i;
        char c;
    }
    Test *tp; //defines pointer to Test struct
    Test t;
    tp = new Test;
    assert (to !=NULL);
    tp->i = 10//Call member function using arrow -> operator
    tp->c='v';

    *tp = t;
    delete tp;
}
```