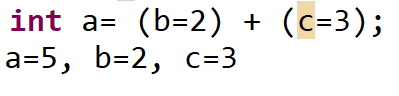
a= b+++c is a= (b++)+c we take b, then add 1 to b, then add c to pre b, then put in a. but a = b+ ++c is a=b+(++c)

* 

**גדלי משתנים בבתים כל בית הוא 8 ביט:**

**char: 1; short: 2; int: 4; long: 4/8; unsigned (int): 4; float: 4; double: 8;**

**long double: 8\16;**

* sizeof() יחזיר תשובה בבתים.
* char מכיל מספרים בין 0 ל-255 ומשתמש ב-ASCII.
* 0x – התחלה של הקסאדצימלי, 0 התחלה של אוקטלי.

typedef vs. #defineדוגמא לשימוש: "typedef unsigned long size\_t;"

1. typedef מוגבל למתן שמות לטיפוסים בלבד, כאשר #define יכול להגדיר שמות נוספים גם עבור ערכים.
2. העיבוד של typedef נעשה ע"י הקומפיילר, כאשר העיבוד של #define מבוצע ע"י ה-preprocessor.
3. typedef הוא הגדרה ממשית של טיפוס חדש, כאשר #define יבצע פשוט הדבקה של הערכים בכל המקומות בהם הם משומשים.
4. typedef יבוצע לפי ה-"scope rule", כלומר יהיה רלוונטי רק ב-scope בו הוגדר (בתוך הפונקציה). ו-#define יוחלף בכל ההופעות לאחר ההגדרה, בלי קשר ל-scope ע"י ה-preprocessor.

**סוגי אחסון של משתנים**

* משתנים לוקאליים – יושבים במחסנית של הקריאות ה-activation record, מופיעים ונעלמים ע"פ קריאות של פונקציה. הטווח שלהם הוא הבלוק בו הם הוגדרו.
* משתנים גלובאליים –לא משויכים לשום פונקציה. יושבים ביניהן. ה-scope שלהם הוא מנקודת ההגדרה עד סוף הקובץ. הם יושבים ב-static memory. עדיף להשתמש בהם כמה שפחות כי אז מאבדים שליטה. נרצה עבודה לוקאלית ושתהיה לנו שליטה מלאה עליה. נשתמש למקרים מיוחדים (למשל מונה, שם התוכנה). חי למשך כל הריצה.
* extern – יש גישה למשתנה הזה מכמה קבצים. הקומפיילר לא שומר זיכרון אלא בזמן ה-linkage מישהו אחר צריך להגדיר אותו ופונים אליו.ברירות מחדל 0. זה הברירת מחדל למשתנה מאותחל מחוץ לבלוק.
* static – גם קיים בכל זמן הריצה (כמו גלובאלי) אבל ה-scope שלו הוא רק הפונקציה שבתוכה אנחנו נמצאים (כמו לוקאלי). מאותחל פעם אחת. ברירת המחדל היא אתחול ל-0. משתנה סטטי בתוך פונקציה, הערך שלו נשמר מפעם לפעם, אפילו אם נראה כאילו הוגדר מחדש.
* static external – מגביל את הטווח של משתנה גלובאלי לקובץ הנוכחי בלבד.
* register - "המלצה" לקומפיילר לשים את המשתנה בתוך רגיסטר.
* auto- הסוג של משתנים מוגדרים בתוך בלוק- זה ברירת מחדל הכוונה.
* משתנים של פונקציה נשמרים על המחסנית ונמחקים בסוף קריאה לפונקציה, אלה אם כן משתנה שהוקצה ואז זה נשמר על הheap ונשמר ביציאה מן הפונקציה.פונקציה static - כמו private בג'אווה – אפשר לקרוא לה רק מאותו הקובץ.

**מערכים** sizeof(arr) = how many bytes does the array take

sizeof(arr)/sizeof(arr[0]) = number of elements in the array.

char d[]="abc" //same as char d[] = {'a', 'b', 'c', '\0'}

שמדפיסים מחורזת זה מדפיס ממיקום המצביע עד לnull ואז מפסיק.

אורך מחרוזת היא 3 אבל תופס 4 בתים

* מערכים מסוג static ו-external מאותחלים ל-0 כברירת מחדל.
* int a[3] ={0} שקול ל: int a[3] ={0,0,0} , ואם היה 1 במום 0 היה מאתחל ל-1 ואז אפסים
* **מערך- ובפרט מחרוזת הוא כמו מצביע שאי אפשר לעשות השמה מחדש של המצביע אם בכתיב של [] ולא \***.

**מצביעים**

* מצביע תופס 4/8 בתים (תלוי במחשב)

p=0; // p doesn't point to any variable. Same as p= NULL;

p = (int\*) 1234; // wrong use. tells p to point to address 1234

אריתמטיקה בין מצביעים:

* חיסור בין מצביעים = כמות (ב-int) האלמנטים בין שניהם.
* (int)p = למשל אם הכתובת היא F אז יחזור לי 15.
* a[j-1] שקול ל-\*(a+j-1). לדוגמא בפוינטר השקילות היא: p[3]=\*(p+3)

מצביע כללי:

נניח שיש לי מצביע שאני לא יודעת למה הוא יצביע (אני רוצה **מצביע כללי**): נגדיר void \*p – נותן להצביע למה שאני רוצה.

ואז אני לא יכולה לעשות \*p – כלומר dereference לא תקין במצביע כללי כי הקומפיילר לא יודע מה יש שם.

מה שכן אפשר לעשות זה להשתמש בקאסטינג: \*((int\*)p) – הופכים למצביע ל-int ואז כן אפשר לגשת למה שיש בו. אבל – זה על אחריותי. אם יש שם double קורה דברים לא צפויים.

הקצאת מערכים דינמית:

Malloc starts elements with trash and its input is the number of bytes needed. Calloc starts elements with 0’s and his input is the number of elements and the size of each element.

* void \***realloc**(void\* ptr, size\_t size): מקבלת מצביע וגודל, ומרחיבה או מצמצמת את המצביע לגודל הזה. הזיכרון לפני הקריאה נשאר אותו הדבר, ואם הגודל גדל אז הזיכרון החדש לא מאותחל. אם אין מקום להרחיב את הזיכרון במקום המקורי, אז הזיכרון הקיים יועתק.

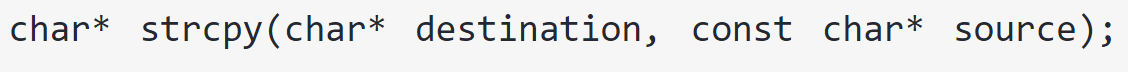
קבועים - const לפני משתנה אומר שערך המשתנה הוא קבוע.

**const int \*p** – זה אומר ש-p מצביע למשתנה קבוע- אי אפשר לשנות תוכן.

**int\* const p** - אומר שהמצביע עצמו הוא קבוע. אני לא יכולה לשנות את ההצבעה שלו לכתובת אחרת בדיוק כמו מערך.

השוואת מחרוזות: int strcmp(const char \*s1, const char \*s2) מחזירה סדר לקסיקוגרפי ביניהם. אם הן זהות זה יחזיר 0 false, אחרת מספר שונה מ-0 true.

העתקת מחרוזת: מעתיק כולל null אם יש מקום- אין מקום התנהגות לא צפויה.



גודל מחרוזת: strlen מחזיר את אורך המחרוזת בלי '\0', sizeof מחזיר עם.

המרה ממחרוזת לint: int atoi(const char\* str)

שרשור מחרוזות: 

מעתיק את השני לראשון אם יש מקום וגם את ה/0. ואז מחזיר את התוצאה גם.

מערכים רב ממדיים:

* הגדרת המערך בעזרת calloc\malloc לא מושיב את המערך הדו-ממדי ברצף בזיכרון (בניגוד להגדרה כ- int arr[3][4]למשל).

דוגמאות:

* int \*a[5] – מערך של 5 פוינטרים (שיכולים להצביע למערכים באורכים שונים)
* int (\*a)[5] – מצביע למערך שכל תא מצביע למערך בגודל 5.

ארגומנטים של main: argc = מס' הארגומנטים (מס' המילים השלמות!) + 1 לשם התכנית. הארגומנט הראשון הוא תמיד שם התכנית.

**מצביעים לפונקציות (ופונקציות כארגומנטים)**

בשביל להגדיר מצביע לפונקציה צריך ערך החזרה, שם וארגומנטים.

דוגמא להגדרת המשתנה בשם f שמצביע לפונקציה עם החתימה שמקבלת int ומחזירה int: **int (\*f)(int)**, ואז שימוש בפונקציה: **(\*f)(x)**.

**Structures**

חיבור מספר משתנים (אין פונקציות בתוך ה-struct!), כל ה-structs הם public.

אתחול – external\static מאותחלים ל-0, אחרת מזובלים ואני חייבת להגדיר.

אופרטור חץ: . 

העברה לפונקציות: structs מועתקים לפונקציה במלואם. לכן עדיף להעביר מצביע ל-struct. נגיד מעבירים struct בלי מצביע זה גם לוקח יותר זמן ואי אפשר יהיה לשנות משתנים שהם "immutable". כלומר נוכל לשנות תוכן של מערכים בפנים אבל אל לאן המערך מבציע נגיד. אבל זה רק אם המערך מסוג מצביע ולא מסוג [], כי המערך מועתק איבר איבר.

Union: ההבדל הוא שכל השדות תופסים את אותו המקום. אותם ביטים משמשים למידע בכל השדות. למעשה גודל השדה יהיה כגודל השדה הגדול ביותר. כביכול כל שדה מסתכל על אותו רצף ביטים ומפרש אותו אחרת.

**CUDA - Parallel Programming**

חומרה = GPU. תכנות עם מעבדים רבים ותהליכים רבים שרצים במקביל.

הארכיטקטורה - יש לנו CPU (המוכר והקיים) ו-GPU (החדש) ולכל אחד מהם יש זיכרון. מי שמנהל את הכול זה ה-CPU: כשנרצה להפעיל אלגוריתמים מקביליים נעשה העתקה מה-CPU ל-GPU. ואז ירוץ kernel – פיסת קוד שמבצעת את העבודה המקבילית. אחרי שהיא תבצע עבודה היא תבצע שינוי בזיכרון ואז תהיה העתקה של הזיכרון לשל ה-CPU.

thread = one independent path of execution through the code.

* בכל בלוק עד 1024 threads שירוצו בו זמנית.

כל בלוק נראה אותו הדבר – כל חלוקה שלו ל-threads היא אותה חלוקה.

ב-GPU יש 3 סוגי זכרונות:

1. **זיכרון לוקאלי** – בתוך ה-kernel יכולים להיות משתנים לוקאליים. שלכל thread שרץ יהיה את הזיכרון הלוקאלי שלו.
2. **זיכרון גלובאלי** – מה שהעתקנו מה-CPU ל-GPU. כולם רואים את אותו הזיכרון: גם ה-GPU וגם ה-CPU.
3. **זיכרון משותף**. בין לבין. זה זיכרון משותף לכל ה-threads. אם הם צריכים לתקשר אחד עם השני.

|  |  |
| --- | --- |
| 1. קריאה לפונקציה ב-GPU. מעבירים אליו את השליטה. מעבירים את האלמנטים שהזיכרון שלהם שמור ב-GPU! מה שנמצא ב-<<<>>> אלה פרמטרים בשביל הריצה עצמה. המספר a הוא כמה בלוקים יש לי. b הוא מספר ה-threads בבלוק.ירוצו לי a\*b threads. הפקודה הזאת אומרת ל-GPU להריץ את הפונקציה הזאת אבל במקביל. | func <<<a,b>>>(  func parameters) |

סנכרון – במידה והפעולות שרצות במקביל תלויות אחת בשניה, נשתמש בסינכרון:

\_\_syncthreads() - מילה שמורה שאומרת למחשב לעצור, לחכות שכולם יסיימו ורק אז להמשיך בקוד.

\_\_global\_\_ רצה על ה-device (GPU), נקראת ע"י ה-host (CPU)

threadIdx.x גישה לאינדקס של התהליך, blockIdx.x מאפשר גישה לאינדקס של הבלוק.

**פקודות preprocessor (#)**

פקודות שאנחנו יכולים לכתוב בקוד שלנו שירוצו לפני שהקוד מתקמפל בפועל.

#include: הקומפיילר מעתיק את התוכן של הקובץ ב-"" או ב-<> לקוד עצמו.

#define: לקבועים. למשל define PI 3.14159 אומר לו לעבור על כל הקובץ הנוכחי, לחפש כל מקום שכתוב בו PI ולהחליף שם ל-3.14.

#undef: מפה ומטה הוא יפסיק ולא יחליף את זה. (#undef PI).

MACRO – הגרת define כ"פונקציה". ה-MACRO הזה יכול לקבל ארגומנטים. זה ממש נראה כמו פונקציה. זה לא ארגומנט שהוא משתנה אלא טקסט מבחינתנו. זו החלפת מחרוזות: לוקחת את מה שכתוב בתוך הסוגריים. macro: מעתיק את התוכן אחד לאחד. לכן משנה ערכי משתנים וכו'.. **לא רלוונטי** פה העניין של העברה לפונקציה by-value. בנוסף מעגל ל-int.

קיימים גם MARCO מובנים של date, file, line, time.

בעיות במאקרו: 1. לא צריך ואף בעייתי לשים **;** בסוף מאקרו. 2. רווח בין "שם הפונקציה" ל"ארגומנטים" – פשוט מחליף איפה שנמצא שם הפונקציה למחרוזת שאחריה.

Conditional Compilation: הכנסת קוד רק אם מתקיימים תנאים מסוימים.

רק מה שבין ה-if ל-endif יעבוד.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone screen with text

Description automatically generated

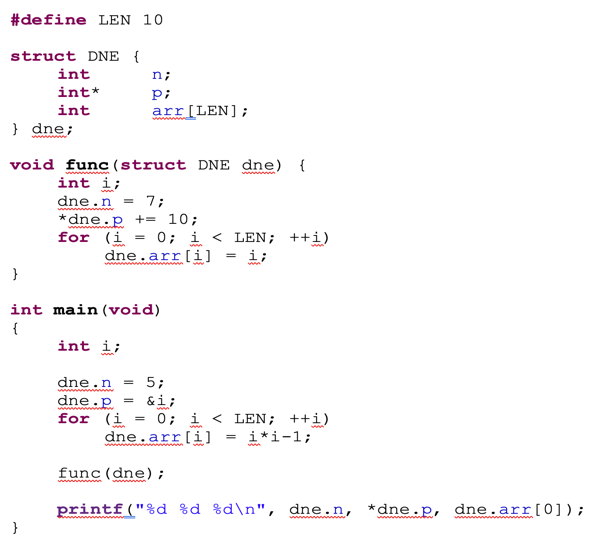
-משתנים מקומי דורש משתנה שיותר גלובלי! להבין על איזה משתנה מדובר בכל פקודה! - אם במקרו אין סוגריים לא להוסיף אותם בראש!! חייבים רווח בין קלט לגוף ואסור רווח בין שם לתחילת קלט.

8/4

4\*5

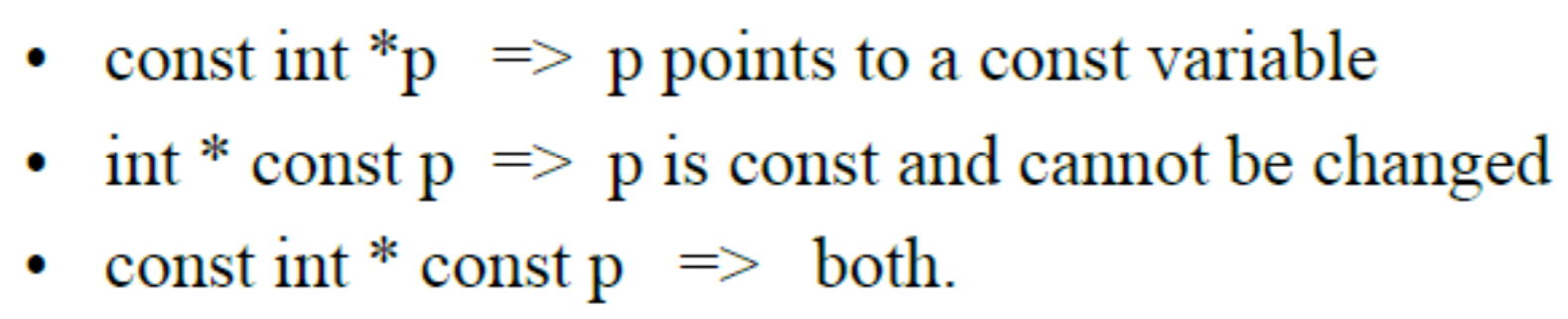
A screenshot of a cell phone

Description automatically generated



5,20,-1

When sending struct to a functionit does a “deap copy”- ie: copies each element to a new struct and does the changes there. But &p is a pointer so it changes, if int\*arr not arr[len] would have changed array as well (because pointer)

****

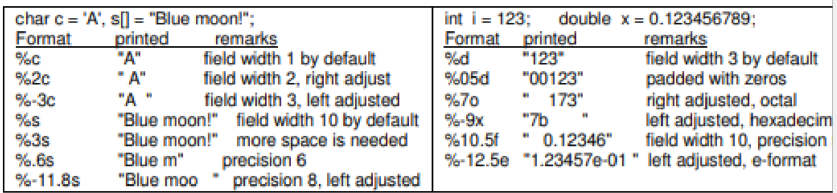
-מדפיס את \*ptr ואז מקדם את ptr ב-1

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated -מדפיס את \*ptr ואז מגדיל את הערך שבתוך ptr ב-1

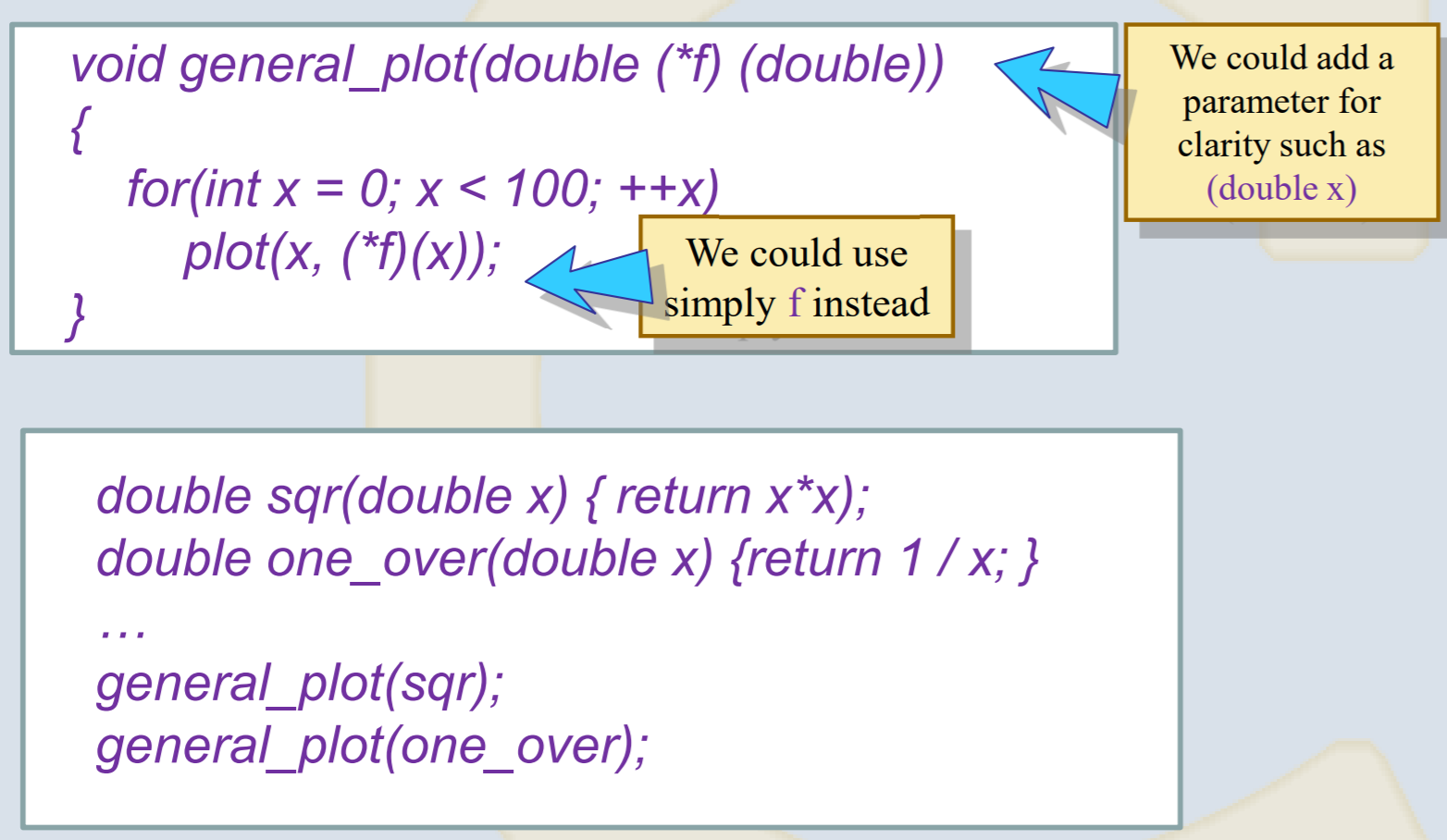
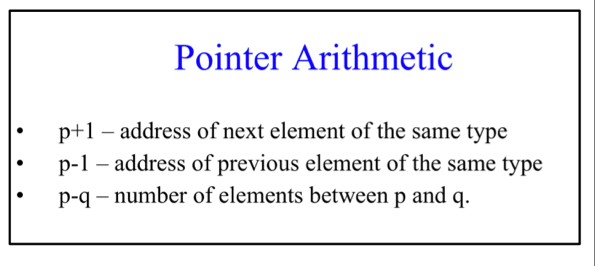
- מקדם את ptr במקום 1 ואז מדפיס את הערך החדש



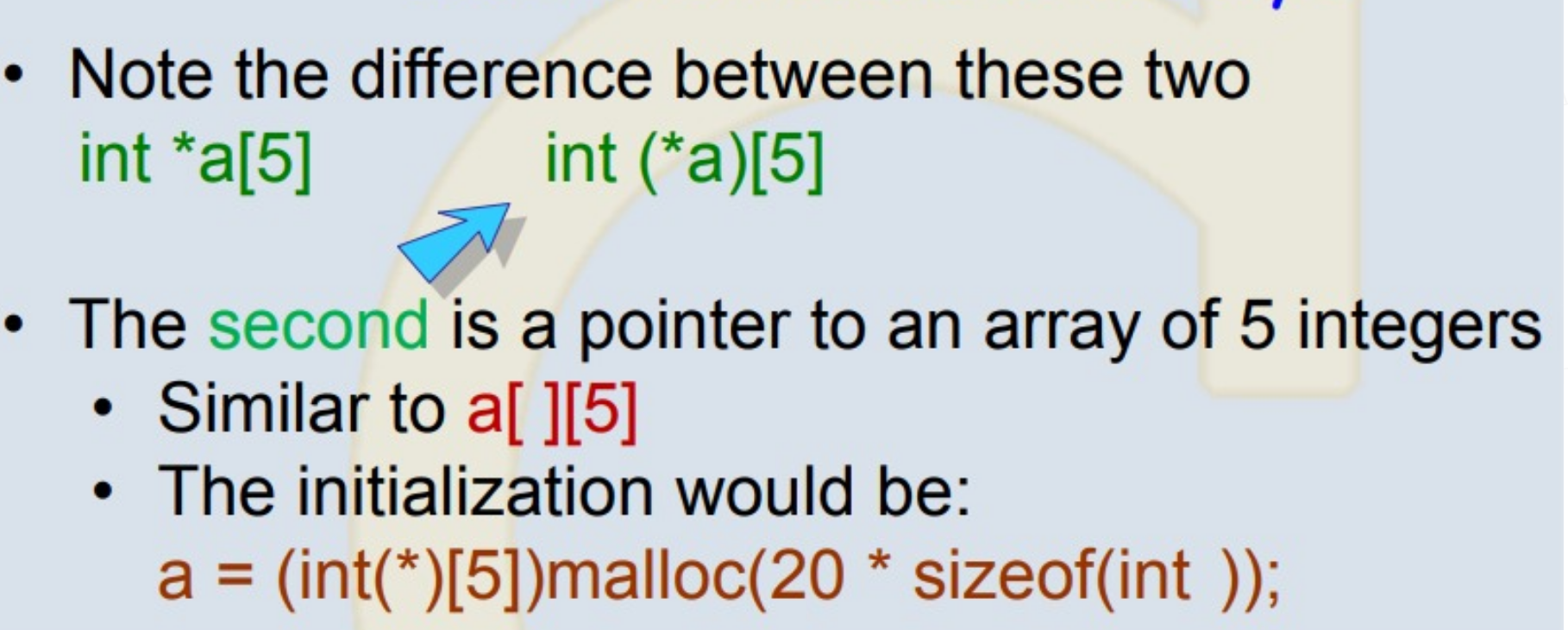
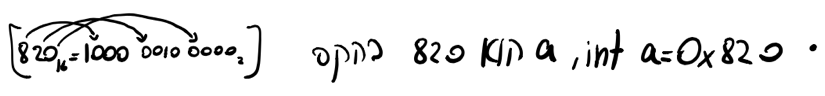
Top first bottom last

A picture containing night, dark, lit, sitting

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Int a =1<<n

Undefined if n>=32 or n<0, works but does nonsense

A++ takes a and the adds 1, ++A adds 1 to a and then takes A

Extern int a- looks for global a from any file. If not found linker error.

|  |  |
| --- | --- |
| Str s[] = “a” | Str\* s = “a” |
| Dies out of function because saved on stack | If defined in function is ok to return |
| Mutable bur cannot change where s points to. | Imutable- s[0] = ‘b’ **runtime**. it can change where s points to |

Strlen(null) is undefined, strlen(p) when p points to null is runtime.

Free(null) is ok and does nothin. Free(p) where p points to null is ok.

A screenshot of a cell phone

Description automatically generated

code works, string saved in special place not stack. If was chrar s[] = “a” would not work because then would be saved on the stack and return pointer to trash. If value was dynamically saved always works, if not dynamic and not קבוע מחרוזת and not global var will return trash. (undefind behavior if used)