OEFENINGEN POINTERS

1. Schrijf een stuk code die een getal tot de derde macht zet (dus x = x \* x \* x).  
Doe het met twee methodes, waarbij de eerste pass-by-value werkt en de tweede pass-by-reference.  
Hint: als je die twee begrippen niet kent, dan even googlen. Snap je het nog niet, dan gewoon meteen de uitwerking bekijken. Let wel: dit zijn heeeeeele belangrijke zaken om te kennen, dus zorg dat je het snapt!

2. Schrijf een stuk code die, door het gebruik van pointers, de inhoud van een array omzet naar hoofdletters. Tip: gebruik de C bibliotheek ctype.h om de functie *toupper* te kunnen gebruiken.

3. Schrijf een stuk code die een array afdrukt, maar zorg er voor dat de data waar naar verwezen wordt niet gewijzigd kan worden. Tip: gebruik *const* om naar de array te verwijzen. Als dit te moeilijk klinkt, dan gewoon de uitwerking bekijken (en begrijpen!).

4. We hebben bij de arrays al een bubble-sort sorteer routine gebruikt (als je dat schrijft als: bubble-sortsorteerroutine, dan heb je in Scrabble 9 keer woordwaarde!). Maak nu eens een bubble-sort, maar dan met pass-by-reference. Tip: gebruik twee methodes: de bubble-sort en de verwissel methode.

5. Bekijk de volgende code:  
// Met het commando sizeof het aantal  
// bytes in een array uitlezen

#include <stdio.h>  
#define GROOTTE 20

size\_t getSize(float \*ptr); // prototype

int main(void)  
{  
 float array[GROOTTE]; // create array

printf("Het aantal bytes van deze array is %u"   
 "\nHet aantal bytes met GetSize is %u\n",  
 sizeof(array), getSize(array));   
}

// grootte van de pointer geven  
size\_t getSize(float \*ptr)  
{  
 return sizeof(ptr);  
}

Kun je een verklaring geven waarom de waarden van sizeof en getSize zo zijn?

6. Even voor de leuk: schrijf een stukje code die de grootte (in bytes) van alle data typen laat zien. Tip: gebruik wederom *sizeof*.

7. Als laatste: geen code die je zelf moet schrijven, maar code die je goed moet bekijken. Hierin staan de manieren waarop je met arrays en pointers kunt verwiijzen. Dan zie je de verschillen (en hopelijk ook de overeenkomsten).

// Indexen en pointers met array manieren.

#include <stdio.h>

#define ARRAY\_GROOTTE 4

int main(void)

{

int b[] = {10, 20, 30, 40}; // maak array en vullen

int \*bPtr = b; // pointer bPtr wijst naar array b

// toon array b met array index notatie

puts("Array b wordt hier getoond met:\narray index notation");

// lus van array b

for (size\_t i = 0; i < ARRAY\_GROOTTE; ++i) {

printf("b[%u] = %d\n", i, b[i]);

}

// array b m.b.v. array naam en pointer/offset notatie

puts("\nPointer/offset notatie waar\n"

"de pointer de array naam is");

// lus door array b

for (size\_t offset = 0; offset < ARRAY\_GROOTTE; ++offset) {

printf("\*(b + %u) = %d\n", offset, \*(b + offset));

}

//de offset zit hem in: \*(array + offset)

// toon array b met bPtr en array index notatie

puts("\nPointer index notatie");

// lus door array b

for (size\_t i = 0; i < ARRAY\_GROOTTE; ++i) {

printf("bPtr[%u] = %d\n", i, bPtr[i]);

}

//de index notatie zit hem in: pointer[index]

// toon array b met bPtr en pointer/offset notatie

puts("\nPointer/offset notatie");

// weer door array b heenfietsen

for (size\_t offset = 0; offset < ARRAY\_GROOTTE; ++offset) {

printf("\*(bPtr + %u) = %d\n", offset, \*(bPtr + offset));

}

//de pointer/offset notatie ziet er uit als: \*(pointer+offset)

}

1. Schrijf een stuk code die een getal tot de derde macht zet (dus x = x \* x \* x).  
Doe het met twee methodes, waarbij de eerste pass-by-value werkt en de tweede pass-by-reference.

// Variabele tot de derde macht met pass-by-value en pass-by-reference.

#include <stdio.h>

int cubeByValue(int n); // prototype v.d. functie pass-by-value

void cubeByReference(int \*gPtr); // prototype pass-by-reference

int main(void)

{

int getal = 5; // getal initialiseren

printf("De originele waarde van getal is: %d", getal);

// geef WAARDE van getal door aan functie cubeByValue

getal = cubeByValue(getal);

printf("\nDe waarde van getal is nu geworden: %d\n", getal);

printf("\n\nEn dan nu met de andere manier (pass-by-reference…)\n\n");

getal=5; //weer even op oude waarde zetten.

printf("De originele waarde van getal is: %d", getal);

// geef ADRES van getal door aan cubeByReference

cubeByReference(&getal);

printf("\nDe waarde van getal is nu geworden: %d\n", getal);

printf("\nZie je het verschil? Nou? Nou? Nou?\n");

}

// bereken en geef terug de derde macht van een integer

int cubeByValue(int n)

{

return n \* n \* n; // derde macht en teruggeven aan de aanroeper van de functie

}

// bereken derde macht via \*gPtr; dit veranderd dus getal in de main functie

void cubeByReference(int \*gPtr)

{

\*gPtr = \*gPtr \* \*gPtr \* \*gPtr; // derde macht \*gPtr

}

2. Schrijf een stuk code die, door het gebruik van pointers, de inhoud van een array omzet naar hoofdletters.

// Omzetten string naar hoofdletters

// Dit is een voorbeeld van een non-constant pointer naar non-constant data.

#include <stdio.h>

#include <ctype.h>

void omzettenNaarHoofdletters(char \*sPtr); // functie prototype

int main(void)

{

char string[] = "kAraKTerS en $03.14"; //char array vullen

printf("De string voor het omzetten: %s", string);

omzettenNaarHoofdletters(string);

printf("\nDe string na het omzetten is nu: %s\n", string);

}

// Omzetten string naar hoofdletters

void omzettenNaarHoofdletters(char \*sPtr)

{

while (\*sPtr != '\0') { // huidige karakter is geen '\0' (einde string)

\*sPtr = toupper(\*sPtr); // Omzetten naar hoofdletters (makkelijk nietwaar?)

++sPtr; // sPtr naar volgende karakter laten wijzen

}

}

3. Schrijf een stuk code die een array afdrukt, maar zorg er voor dat de data waar naar verwezen wordt niet gewijzigd kan worden.

// Afdrukken van een string, karakter voor karakter

// Voorbeeld van non-constant pointer naar constant data.

#include <stdio.h>

void drukKaraktersAf(const char \*sPtr); //pointer naar constante data

int main(void)

{

char string[] = "Nou, toon deze karakters dan maar eens.";

puts("De string is:");

drukKaraktersAf(string);

puts("");

}

// sPtr kan nu niet gebruikt worden om de karakters waar die naar wijst te zijzigen,

// nu is sPtr een "read-only" pointer

void drukKaraktersAf(const char \*sPtr)

{

// door de hele string wandelen

for (; \*sPtr != '\0'; ++sPtr) { // geen initialisatie van sPtr

printf("%c", \*sPtr);

}

}

4. Maak nu eens een bubble-sort, maar dan met pass-by-reference.

// Zet waarden in een array, sorteer deze waarden

// in oplopende volgorde en laat het zien

#include <stdio.h>

#define GROOTTE 10

void bubbleSort(int \* const array, const size\_t lengte);

int main(void)

{

// array vullen

int a[GROOTTE] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

puts("Data in array in originle volgorde: ");

// door a heenwandelen

for (size\_t i = 0; i < GROOTTE; ++i) {

printf("%4d", a[i]);

}

bubbleSort(a, GROOTTE); // sorteren van de array

puts("\nEn nu in oplopende volgorde gesorteerd: ");

for (size\_t i = 0; i < GROOTTE; ++i) {

printf("%4d", a[i]);

}

puts("");

}

// sorteer een array van integers met een bubble-sort

void bubbleSort(int \* const array, const size\_t lengte)

{

void verwissel(int \*element1Ptr, int \*element2Ptr); // prototype

// loop om aantal keer doorloop te controleren

for (unsigned int lus = 0; lus < lengte - 1; ++lus) {

// lus om de controle op twee getallen te controleren

for (size\_t j = 0; j < lengte - 1; ++j) {

// verwissel twee getallen als ze niet in goede volgorde liggen

if (array[j] > array[j + 1]) {

verwissel(&array[j], &array[j + 1]);

}

}

}

}

// verwissel waardes op geheugenlocatie waar element1Ptr en

// element2Ptr naar toe wijzen

void verwissel(int \*element1Ptr, int \*element2Ptr)

{

int tussenWaarde = \*element1Ptr;

\*element1Ptr = \*element2Ptr;

\*element2Ptr = tussenWaarde;

}

5. Het commando *sizeof* wordt gebruikt om het aantal bytes in een array (of een ander data type!) weer te geven. In deze code wordt *sizeof* toegepast op een array van 20 ‘dingen’ de het data type float hebben. Float variabelen worden (vaak/meestal/hopelijk) weergegeven met 4 bytes. Dus dan wordt het 20 elementen \* 4 bytes = 80 bytes.

Even een zijstapje:  
als je bijvoorbeeld een array hebt als: double real[44];  
en je weet dat een real 8 bytes lang is, dan weet je dat de array 44\*8 = 352 bytes groot is.  
Als je wilt weten hoeveel elementen er in die array zitten (nu weten we dat, maar stel dat je dat niet weet), dan kun je sizeof(real) / sizeof(real[0]) gebruiken.

Okay, terug naar de methode getSize; deze krijgt een array van 20 elementen op zijn bordje, en de parameter van die functie is een pointer naar het eerste element van de array ( a[0] ).  
Als ik *sizeof* met een pointer gebruik, dan komt ‘ie terug met de grootte van de pointer en niet met de grootte van het element waar die naar wijst.  
Een pointer is meestal 8 bytes (of groter), dus getSize komt terug met 8.

6. Schrijf een stukje code die de grootte (in bytes) van alle data typen laat zien. Tip: gebruik wederom *sizeof*

// Met sizeof alle standaard data type groottes tonen.

#include <stdio.h>

int main(void)

{

char c;

short s;

int i;

long l;

long long ll;

float f;

double d;

long double ld;

int array[20]; // array van 20 int elementen

int \*ptr = array; // pointer naar

printf(" sizeof c = %u\tsizeof(char) = %u"

"\n sizeof s = %u\tsizeof(short) = %u"

"\n sizeof i = %u\tsizeof(int) = %u"

"\n sizeof l = %u\tsizeof(long) = %u"

"\n sizeof ll = %u\tsizeof(long long) = %u"

"\n sizeof f = %u\tsizeof(float) = %u"

"\n sizeof d = %u\tsizeof(double) = %u"

"\n sizeof ld = %u\tsizeof(long double) = %u"

"\n sizeof array = %u"

"\n sizeof ptr = %u\n",

sizeof c, sizeof(char), sizeof s, sizeof(short), sizeof i,

sizeof(int), sizeof l, sizeof(long), sizeof ll,

sizeof(long long), sizeof f, sizeof(float), sizeof d,

sizeof(double), sizeof ld, sizeof(long double),

sizeof array, sizeof ptr);

}

.