Pointeri

Pointeri

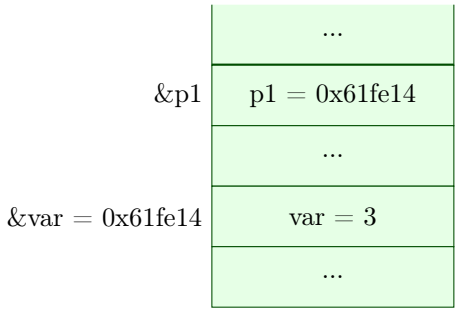
O variabilă este un container folosit pentru a stoca date la o adresă asignată automat.

Pointerul este o variabilă care stochează adresa unei alte variabile. Operația prin care se obține valoarea stocată la adresa salvată în pointer se numește dereferențiere.

Rulați următoarea secvență de cod:

```
int main() {
    int var = 3;
    int *p1;
    p1 = &var;

    // Adresa pointerului
    printf("Adresa pointerului (&p1) = %x\n\n", &p1);

    // Adresa variabilei = adresa stocata de pointer
    printf("Adresa variabilei (&var) = %x\n", &var);
    printf("Adresa stocata de pointer (p1) = %x\n\n", p1);

    // Valoarea variabilei = derefentierea pointerului
    printf("Valoarea variabilei (var): %d\n", var);
    printf("Dereferentierea pointerului (*p1): = %d \n", *p1);

    return 0;
}
```

Presupunând că adresa asignată lui `var` este `0x61fe14`, stiva va arăta astfel:



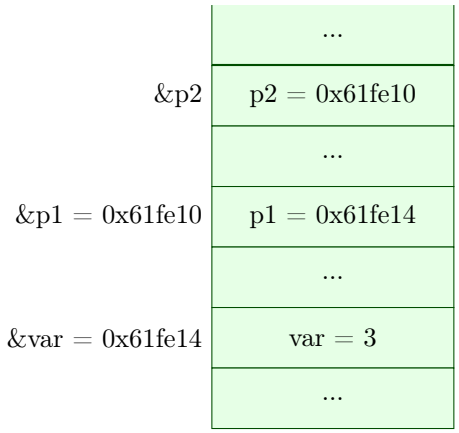
Stack

Un pointer dublu este un pointer care stochează adresa unui alt pointer. Analog pentru un pointer-`n`. Rulați următoarea secvență de cod:

```
int main() {
    int var = 3;
    int *p1;
    p1 = &var;
```

```
    int **p2;
    p2 = &p1;

    // Adresa pointerului dublu
    printf("Adresa pointerului dublu (&p2) = %x\n\n", &p2);

    // Adresa pointerului simplu = adresa stocata de poinerul dublu
    printf("Adresa pointerului simplu (&p1) = %x\n", &p1);
    printf("Adresa stocata de pointerul dublu (p2) = %x\n\n", p2);

    // Adresa variabilei = adresa stocata de pointerul simplu
    printf("Adresa variabilei (&var) = %x\n", &var);
    printf("Adresa stocata de pointerul simplu (p1) = %x\n", p1);
    printf("Prima dereferntiere a pointerului dublu (*p2) = %x\n\n", *p2);

    // Valoarea variabilei = derefentierea pointerilor simplu/dublu
    printf("Valoarea variabilei (var) = %d\n", var);
    printf("Derefentierea pointerului simplu (*p1) = %d\n", *p1);
    printf("Dubla dereferentiere a pointerului dublu (**p2) = %d\n", **p2);

    return 0;
}
```

Presupunând că adresa asignată lui `var` este `0x61fe14`, iar adresa care asignată lui `p1` este `0x61fe10`, stiva va arăta astfel:



Stack

Pointeri și tablouri

Atunci când se declară o variabilă de tip tablou, numele său este echivalent un pointer constant către adresa primului element din tablou. Variabila în sine nu este un pointer, dar se comportă ca și cum ar fi. Regula care face posibilă această echivalență se numește *decay to pointer*.

Există doar trei excepții, situații în care variabila se comportă conform tipului ei, nu ca un pointer, și anume: utilizarea variabilei împreună cu operatorul `sizeof`, cu operatorul referință (`&`), și cazul în care variabila este un tablou de caractere inițializat cu valoare constantă (folosind `""`).

În exemplul de mai jos se afișează dimensiunea tabloului `a`, iar rezultatul este `40`, anume spațiul pentru 10 elemente de tip `int`, a câte 4 biți fiecare. Dacă `sizeof` nu ar constitui o excepție de la *decay to pointer*, rezultatul ar fi dimensiunea unui pointer la `int` și ar fi destul de nefolositor.

```
int a[10];
printf("sizeof a: %zu \n", sizeof(a));
```

sizeof a: 40

Însă o dată ce un tablou a fost echivalat cu un pointer (s-a produs *decay to pointer*), nu se mai poate reveni la tablou, variabila trebuie tratată ca pointer. Această situație se întâmplă, cel mai adesea, când un tablou este transmis într-o funcție, ca în exemplul de mai jos.

Exemplul 1: Decay to pointer

```
void f(int *x){
    printf("sizeof x: %zu", sizeof(x));
}

int main(){
    int v[10];
    printf("sizeof v: %zu", sizeof(v));

    f(v);
}
```

sizeof v: 40
sizeof x: 8

Funcția `f` primește ca parametru un pointer. În `main` este apelată cu tabloul `v`, ceea ce este în regulă, având în vedere că `v` este echivalent cu un pointer. Însă atunci când se încearcă aflarea dimensiunii lui `x` în funcție, rezultatul este 8, anume dimensiunea unui pointer, și nu 40, dimensiunea tabloului.

Datorită faptului că tabloul poate fi folosit ca pointer (mai puțin în excepțiile discutate mai sus), în practică apare rar situația în care este nevoie ca o variabilă să fie un pointer la tablou. Însă acest tip poate apărea în programe ca o consecință a *decay to pointer*: dacă o matrice este transmisă într-o funcție (similar cu felul în care vectorul `v` este transmis în `f` în exemplul anterior, variabila va fi echivalată cu un pointer la tablou).

Echivalența dintre pointeri și tablouri face posibil ca pointerul returnat de funcția `malloc` să fie interpretat ca tablou și folosit ca atare. În exemplul de mai jos este alocat spațiu pentru trei întregi și returnat pointerul `p` către blocul respectiv de memorie. Pointerul poate fi folosit ca un tablou de trei elemente, cu indexarea specifică tablourilor, `[]`.

```
int dim = 3;
int *p =(int*) malloc(dim*sizeof(int));
p[1] = 1234;
printf("p[1] = %d", p[1]);
```

Alocarea dinamică

Reamintim funcțiile pentru alocarea, realocarea/eliberarea spațiului alocat dinamic:

```
void* malloc(size_t dim)
void* calloc(size_t, dim)
void* realloc(void *, size_t)
void* free(void*)
```

Funcțiile pentru alocare de spațiu (`malloc`, `calloc`, `realloc`) returnează un pointer către zona de memorie care a fost alocată. În cazul în care nu s-a putut realiza alocarea, pointerul returnat este `NULL`. Din acest motiv, este necesară testarea valorii acestuia pentru a verifica dacă alocarea s-a realizat cu succes.

Dacă este necesară inițializarea variabilei pentru care se alocă spațiu cu 0 sau `NULL`, se poate folosi `calloc`.

Pentru a realoca spațiul pentru o variabilă, se folosește funcția `realloc`. Aceasta primește ca parametru un pointer către zona de memorie care trebuie realocată și noua dimensiunea necesară. Pointerul trebuie să refere la un bloc de memorie. În cazul în care funcția `realloc` primește ca parametru pointer la `NULL`, aceasta se comportă ca funcția `malloc`.

Aplicație (versiunea 1)

Se consideră următoarea structură:

```
struct Masina {
    char *marca;    /* numele compus se citeste cu underscore: AAA_BBB */
    int pret;
    char numar[8]; /* e.g., IF09BCX, B89FMH*/
};
```

Să cere:

1. Creați o bibliotecă pentru procesarea datelor de tipul structurii `Masina`.
2. (1p) Implementați o funcție pentru citirea datelor de la tastatură pentru o variabila de tip `Masina`. Hint: aveți grijă la scurgerile de memorie, la apel multiplu de citire.
3. (1.5p) Implementați o funcție care alocă dinamic spațiu pentru un vector cu elemente de tip `Masina`. Folosiți funcția de citire pentru fiecare element din vector.
4. (0.5p) Implementați o funcție pentru afișarea în consolă a valorilor câmpurilor unei variabile de tip `Masina` (o înregistrare pe linie), și afișați elementele stocate în vector.
5. (1p) Implementați o funcție care permite setarea câmpului marcă al unei mașini (funcția va primi ca parametru nouă marca și variabila de tip `Masina` de modificat).
6. (2p) Implementați o funcție care realizează o copie profundă a unei variabile de tip `Masina` și o stochează în alta variabilă de acest tip. Testați această funcție.
7. (2p) În main, declarați o matrice (tablou bidimensional) cu dimensiune $n \times n$ (n – citit de la tastatura). Matricea conține elemente de tip `Masina`. Alocați dinamic spațiu pentru matricea de mașini. Folosiți metodele implementate la sub-punctele anterioare pentru a citi/afișa date de la tastatură pentru variabila matrice.
8. (1p) Desenați stack-ul și heap-ul pentru vectorul și matricea de mașini.
9. (1p) Eliberați memoria alocată dinamic pentru vectorul/matricea de tip `Masina`. Ați eliberat corect memoria? Verificați pointerii pe care i-ați folosit pentru alocare.

Extra

Folosind materialul opțional pe tema versionari de cod (Git):

- folosiți Valgrind pentru a verifica dacă programul are scurgeri de memorie.
- faceți citirea vectorului cu elemente de tip `Masina` dintr-un fișier text.
- creați un repository local cu fișierele obținute din sub-punctele anterioare.
- faceți push pe remote.

Aplicație (versiunea 2)

Se consideră următoarea structură:

```
struct Piesa {
    char *denumire; /* numele compus se citeste cu underscore: piesa_aliaj */
    int pret;
    char cod[5];    /* e.g., 12BA*/
};
```

Să cere:

1. Creați o bibliotecă pentru procesarea datelor de tipul structurii `Piesa`.
2. (1p) Implementați o funcție pentru citirea datelor de la tastatură pentru o variabila de tip `Piesa`. Hint: aveți grijă la scurgerile de memorie, la apel multiplu de citire.
3. (1.5p) Implementați o funcție care alocă dinamic pentru un vector cu elemente de tipul `Piesa`. Folosiți funcția de citire pentru fiecare element din vectorul alocat.
4. (0.5p) Implementați o funcție pentru afișarea în consolă a valorilor câmpurilor unei variabile de tip `Piesa` (o înregistrare pe linie), și afișați vectorul.
5. (1p) Implementați o funcție care permite setarea câmpului denumire al unei piese (funcția va primi ca parametru nouă denumire și variabila de tip `Piesa` de modificat).
6. (2p) Implementați o funcție care realizează o copie profundă a unei variabile de tip `Piesa` și o stochează în altă variabilă de acest tip. Testați această funcție.
7. (2p) Creați și testați o funcție care șterge o piesa din vector, de pe o poziție `idx`.
8. (1p) Creați o funcție care șterge duplicatele din vector. În implementarea funcției de ștergere a duplicatelor puteți folosi funcția de ștergere implementată anterior.
9. Eliberați memoria alocată dinamic pentru vectorul de tip `Piesa`. Sigur ați eliberat corect memoria? Verificați toți pointerii pe care i-ați folosit pentru alocare.

Extra

Folosind materialul opțional pe tema versionari de cod (Git):

- folosiți Valgrind pentru a verifica dacă programul are scurgeri de memorie.
- faceți citirea vectorului cu elemente de tip `Piesa` dintr-un fișier text.
- creați un repository local cu fișierele obținute din sub-punctele anterioare.
- faceți push pe remote.

Aplicație (versiunea 3)

Se consideră următoarea structură:

```
struct Stoc {
    char denumire[20]; /* numele compus se citeste cu underscore: stoc_Lidl */
    int nr_prod;       /* numarul de produse aflate in stoc */
    float *pret_prod;  /* vector care stocheaza pretul fiecarui produs */
};
```

Să cere:

1. Creați o bibliotecă pentru procesarea datelor de tipul structurii `Stoc`.
2. (1p) Implementați o funcție pentru citirea datelor de la tastatură pentru o variabilă de tip `Stoc`. Hint: aveți grijă la scurgerile de memorie, la apel multiplu de citire.
3. (1.5p) Implementați o funcție care alocă dinamic un vector de elemente de tipul `Stoc`. Folosiți funcția de citire pentru fiecare element din vectorul alocat.
4. (0.5p) Implementați o funcție pentru afișarea în consolă a valorilor câmpurilor unei variabile de tip `Stoc` (o înregistrare pe linie), și afișați vectorul de tip `Stoc`.
5. (1p) Implementați o funcție care permite setarea câmpului `pret_prod` dintr-o variabilă de tipul `Stoc` (funcția va primi ca parametru noile prețuri și variabila de tip `Stoc` de modificat).
6. (2p) Implementați o funcție care realizează o copie profundă a unei variabile de tip `Stoc` și o stochează în altă variabilă de acest tip. Testați această funcție.
7. (2p) Creați și testați o funcție care șterge o piesa din vectorul de tip `Stoc`, de pe o poziție `idx` (care este dată ca parametru funcției).
8. (1p) Creați o funcție care șterge duplicatele din vectorul de tip `Stoc`. În implementarea funcției de ștergere a duplicatelor puteți folosi funcția de ștergere implementată anterior.
9. (1p) Eliberați memoria alocată dinamic pentru vectorul de `Stoc`. Sigur ați eliberat corect memoria? Verificați pointerii pe care i-ați folosit pentru alocare.

Extra

Folosind materialul opțional pe tema versionari de cod (Git):

- folosiți Valgrind pentru a verifica dacă programul are scurgeri de memorie.
- faceți citirea vectorului cu elemente de tip `Stoc` dintr-un fișier text.
- creați un repository local cu fișierele obținute din sub-punctele anterioare.
- faceți push pe remote.