Звіт

Автор: Момот Р. КІТ-119а

Дата: 09.04.2020

ЛАБОРАТОРНА РОБОТА № 4. ВНУТРІШНЄ ПОДАННЯ ІНТЕГРОВАНИХ СТРУКТУР ДАНИХ

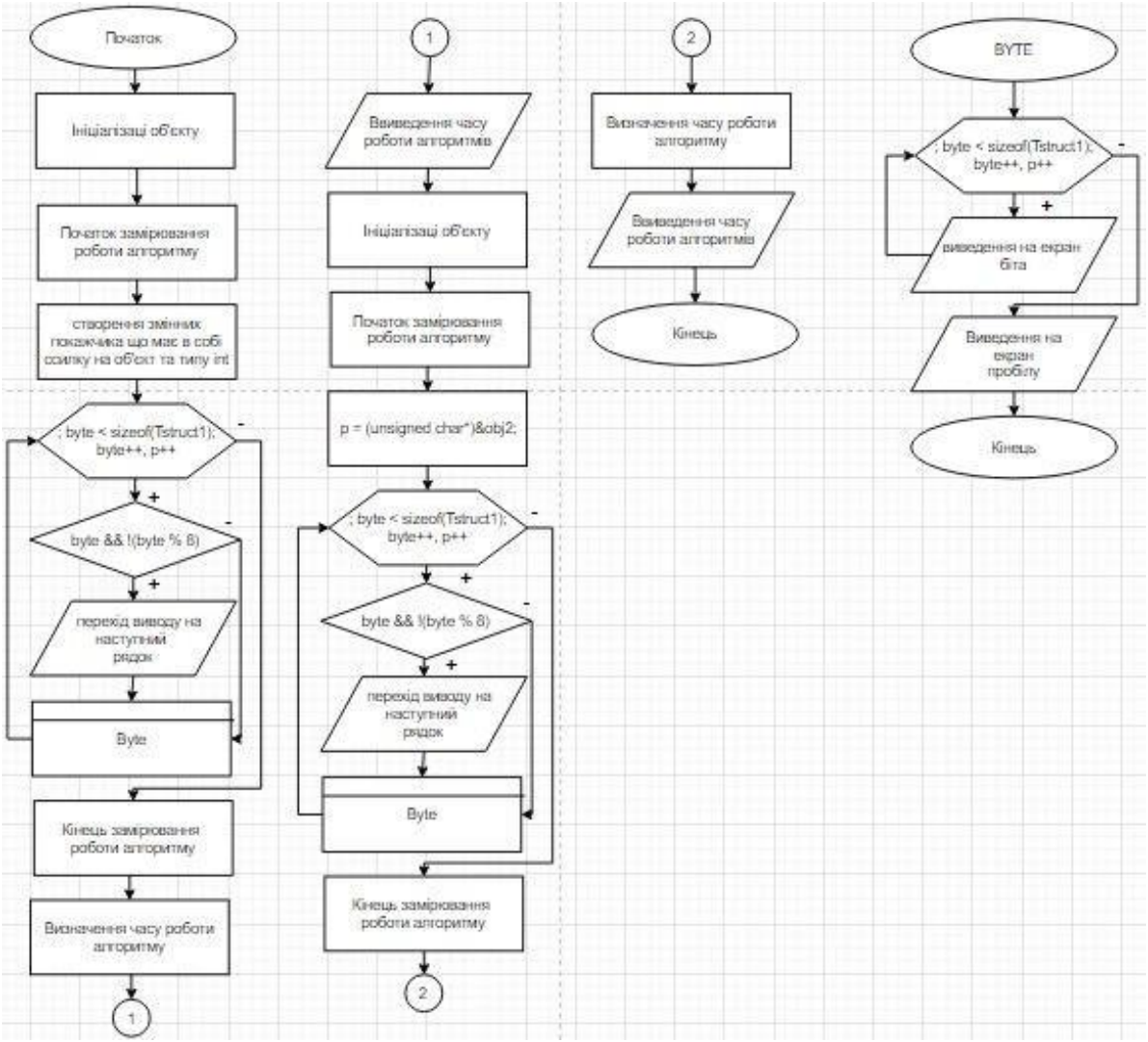
Мета: отримати та закріпити знання про внутрішнє подання інтегрованих структур даних у мовах програмування.

Індивідуальне завдання

Написати програму, яка виводить на екран внутрішнє подання структури з варіантною частиною та з бітовими полями, а також масива структур. Порівняти час доступу до даних з вирівнюванням та за умови відсутності вирівнювання.

Об’єкт	Тип об’єкту	Стан (так/ні)
Дерева	Садові, дикі	-косточкові, -раньої стиглості

Блок-схема алгоритму програми



Текст програми

```
#include <iostream>
#include <conio.h>
#include <chrono>
using namespace std;

void BYTE(unsigned char A)  //виведення вмісту байта
{
    for (int bit = 128; bit >= 1; bit >>= 1)
        cout << (A & bit ? "1" : "0");

    cout << " ";
}

struct Tstruct1
{
    int s;    /* 4 bytes */
    short flip1 : 1;
    short flip2 : 1;
    short flip3 : 1;
    short flip4 : 1;
    short flip5 : 1;
    short flip6 : 4;
    short flip7 : 4;
    bool earlyRipeness;

    union
    {
        struct
        {
            bool kernel;
            double trunkLength;
        } wildTree;

        struct
        {
            short st;
        }gardenTree;

    } un;
};

struct Tstruct2
{
    short s;    /* 4 bytes */
    int flip1;
    int flip2;
    int flip3;
    int flip4;
    int flip5;
    int flip6;
    int flip7;
    bool earlyRipeness;
    bool kernel;
};

void main()
{
    setlocale(LC_ALL, "Rus");
    cout << "Структура с выравниванием" << endl << endl;

    auto begin = chrono::steady_clock::now();

    Tstruct1 obj1 = { 5, 1, 0, 1, 0, 1, 12, 1 };
```

```
    obj1.un.wildTree.trunkLength = 2.5;
    obj1.un.wildTree.kernel = true;
    obj1.earlyRipeness = true;

    unsigned char* p = (unsigned char*)&obj1;
    int byte = 0;
    for (; byte < sizeof(Tstruct1); byte++, p++)
    {
        if (byte && !(byte % 8)) cout << endl;
        BYTE(*p);
    }

    cout << endl << endl;

    auto end = chrono::steady_clock::now();
    auto elapsed_ms = chrono::duration_cast<chrono::nanoseconds>(end - begin);

    cout << "Время с выравниванием: " << elapsed_ms.count() / 10 << "ns" << endl << endl <<
endl;
    cout << "Структура без выравнивания" << endl << endl;

    auto begin2 = chrono::steady_clock::now();

    Tstruct2 obj2 = { 5, 1, 0, 1, 0, 1, 12, 1 };
    obj2.earlyRipeness = true;
    obj2.kernel = true;

    p = (unsigned char*)&obj2;
    for (int byte = 0; byte < sizeof(Tstruct2); byte++, p++)
    {
        if (byte && !(byte % 8)) cout << endl;
        BYTE(*p);
    }

    cout << endl;

    auto end2 = chrono::steady_clock::now();
    auto elapsed_ms2 = chrono::duration_cast<chrono::nanoseconds>(end2 - begin2);

    cout << endl;
    cout << "Время без выравнивания: " << elapsed_ms2.count() / 10 << "ns" << endl;
}
```

Результати роботи програми

```
Структура с выравниванием

00000101 00000000 00000000 00000000 10010101 00000011 00000001 00000000
00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000100 01000000

Время с выравниванием: 1461590ns


Структура без выравнивания

00000101 00000000 11001100 11001100 00000001 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000001 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000001 00000000 00000000 00000000
00001100 00000000 00000000 00000000 00000001 00000000 00000000 00000000
00000001 00000001 00000000 00000000

Время без выравнивания: 2083770ns
```

Висновок

У результаті роботи програми подано внутрішнє (машинне) подання структури з варіантною частиною та з бітовими полями, а також масива структур. Як видно зі скріншоту результата роботи програми, структура із вирівнюванням даних працює швидше, а також затрачує менше пам’яті на зберігання.

Вирівнювання даних впливає на швидкість доступу до них, і як наслідок, на швидкість роботи програми.

Відповіді на питання

1. Чим відрізняються машини з порядком Little Endian від машин Big Endian?

Порядок розміщення бітових болів в байті залежить від порядку байтів. При порядку Little Endian бітові поля розділяються починаючи з перших байтів, при Big Endian бітові поля розділяються починаючи з останніх байтів.

2. Що таке «вірівнювання» і навіщо воно потрібно?

Вирівнювання даних впливає на швидкість виконання програм. Воно полягає в тому, щоб адреса, яка вирівнюється, визначалася як числова адреса по модулю ступеня 2. Загалом вирівнюються в пам'яті поля по межі кратної своєму ж розміру. Якщо ж елемент не вирівняний, то машина сама буде доповнювати його, щоб прочитати машинні слова, а це в свою чергу спричинить те, що машина буде створювати і зчитувати незначущі байти, які і сповільнюють читання, а також займають місця в пам’яті.

3. Як можна змінити вирівнювання даних?

Змінити вирівнювання даних можна за допомогою ключових слів, таких як **_alignof(<тип>)**, яке поверне поточні вимоги вирівнювання для даного типу та **__declspec(align(<число>))**, яке використовується для примусового вирівнювання по типу даних. Також можна використовувати директиви компілятора. Наприклад **#pragmapack(push, <число>)**, яке задає вирівнювання в <число> байт, та **#pragma pack(pop)**, яке повертає попередню настройку.

4. Яким може бути физичне подання структури?

У пам'яті структура може бути представлена в одному з двох видів:

а) у вигляді послідовності полів, які займають безперервну область пам'яті

б) у вигляді зв'язного списку з покажчиками на значення полів структури

5. Як розподіляється пам’ять для структур з варіантними частинами?

Під структуру з варіантами виділяється в будь-якому випадку обсяг пам'яті, достатній для розміщення найбільшого варіанту. Якщо ж виділена пам'ять використовується для меншого варіанту, частина її залишається невикористаної. Загальна для всіх варіантів частина запису розміщується так, щоб зміщення всіх полів щодо початку запису були однаковими для всіх варіантів.

6. Як розподіляється пам’ять для структур з бітовими полями?

Бітове поле, насправді, - це просто особливий тип структури, яка визначає, яку довжину має кожен член. Бітові поля повинні оголошуватись як цілі беззнакові, оскільки 1 біт не може мати знака. Наприклад,

```
#pragma pack (push, 1)
stuct MyBitStruct
{
     uint16_t a : 4; // uint16_t - тип беззнаковий цілий 16 розрядний
     uint16_t b : 4;
     uint16_t c;
};
```

```
#pragma pack (pop)
```

Вийшла структура на 4 байта. Дві половини першого байта - це поля a і b. Другий байт не доступний на ім'я, останні 2 байта доступні на ім'я c.

7. Наведіть приклад доступу до полів структури з варіантної частини.

<Ім'я змінної-запису>. <Ім'я варіанту>. <Ім'я поля>

8. Яке призначення дискриптора структури, хто його створює?

Дискриптор структури потрібен для ідентифікування структури, яка оголошується. Його створює користувач.

9. Чи залежить обсяг пам’ті, що розподіляється для структури, від послідовності полів у її складі?

Ні.

10. Які директиви компілятора призначені для впливу на вирівнювання даних?

#pragmapack(push, <число>), яке задає вирівнювання в <число> байт, та **#pragma pack(pop)**, яке повертає попередню настройку вирівнювання.

11.Чи залежить адреса, яку призначає компілятор даним, від типу цих даних?

Так, адреса, яку призначає компілятор залежить від типу даних.

12. Від чого залежить розмір структури в пам’яті комп’ютера?

Розмір структури в пам’яті комп’ютера залежить від кількості полів у структурі.

13.Як вирівнюються структури у масиві структур?

У випадку використання масиву вирівняних структур даних вирівняною в загальному випадку виявляється тільки перша структура. У наслідок чого компілятор додає додаткові невикористовувані байти у кінець оброблюваної структури.