C++ Управление памятью

Адреса и ссылки

Адрес – номер ячейки оп. памяти, содержащей данные.
правила присвоения адресов определены архитектурой.
операции над адресами реализуют машинные команды
Ссылка – имя, сопоставимое адресу, по которому лежат
данные с изв. типом. Адрес строго постоянен для имени.
операции над ссылками реализует станд. библиотека.
Взятие ссылки – &имя. Предоставляет адрес данных
обеспечивает многократную подготовку адреса.
Разыменование – *имя. Спец. переменная, хранящая в
виде значения адрес данных. Имеет собственный адрес

Понятие указателя

Указатель – та самая переменная/имя или аргумент функции.
- Предполагаемый тип задается при объявлении.
- Void указатель может хранить данные любых типов.
- Тип данных, сопоставляемый указателю, реализует то же множество операций, что и unsigned short
- При доступе к данным не проверяется соответствие пр. имен.
- адрес может быть присвоен указателю напрямую

  int a=5;
  int *b=&a;
- пустой указатель – не хранит адрес
- автоматической проверки на пустоту нет
- Доступ по пустому указателю приводит к ошибке
- Ручной контроль использует Null/nullptr.

  int a=5; – данные
  int *b=&a; – указатель на адрес данных
  int &c=a; – ссылка на данные
  к b нельзя присвоить значения
  к c можно, причем изменится a.

Передача аргументов в функцию

- Копированием значения – данные аргумента копиру-

кота внутрь функции.
- По адресу (&) - аргумент это ссылка на данные, опред во внешней программе.
- Копированием адреса (*) - копируется указатель.

Пример передачи на C++HUB.

Доступ к аргументам по адресу
- Точкой - если аргумент переменная или ссылка, т.
  получим внешнюю таблицу имен.
- Звездочка - доступ через указатель ≈ ссылка
- стрелочка - прямой доступ к таблице имен.

Арифметика адресов

- Имя массива - указатель на первый элемент
- смещение - целое знаковое число, индекс, относительно первого элемента.
- индексация - смещение с учетом размера данных (sizeof)

```
int array[10]            int b = array[5];
int* p_arr = array;      int b = *(p_arr + 5);
```

```
int array[10][5];  двумерный массив
int * array[10];   массив указателей
int** array;       указатель на указатель
```

Статическая/динамическая память

Ст. память - память выделяемая транслятором в м. коде
- адреса выделяемые при загрузке программы, не меняются
- Под данные резервируется объем памяти

Дин. память - вся остальная память выч. машины.
- распределяется между раб. программами на машине
- требует реализации (на уровне stdlib или ОС.)

new <тип> / new <тип>[] - Запрос диапазона памяти под известный тип данных. Возвращает адрес первого байта диапазона. Бросает исключение при неудаче std::runtime_error.

Терминантный массив —
Длина значимой части данных неизвестна.
Конец последовательности заканчивается терминатором
    char str[10] = "str"; // {"s", "t", "r", 0x00}

Указатель на функцию

Адрес функции — присваивается транслятором, либо помещается в таблицу внешних имен. По размеру и машинному представлению не отличается от других адресов.
Указатель на функцию — адрес, записанный в перемен. спец. вида:
    <тип возврата> (*имя) (<типы арг.>);
• Присваивание указателя на функцию аналогично перемен.
• позволяет реализовать "управляемую перегрузку"
• вызов по указателю идет без разыменовывания *.

Функции обратного вызова (Callback)

Обратные вызовы — техника, при которой предоставляется доступ к переменной-указателю на наперед неизв. функцию.
• Запись доступного извне указателя называется регистрац. функции обратного вызова.
• Главная программа определяет только момент вызова.
• Функция не имеет доступа к данным программы.

```
void Register (float (*act) (float, float))
{ action = act; }
float (*action) (float a, float b);
```

Пример указателя на функцию:

```
float add (float a, float b);
float sub (float a, float b);
    float (*action) (float a, float b);
        action = add или sub; // привяз.
        action (a1, a2);      // вызов.
```

Колбэки:
См. объявление выше.
    Инициализация Register (add или sub);

Приведение типов

Операция представления значения известного типа к значению другого известного типа.
В результате получим:
- множество типа, пересечение двух типов
- машинное представление выводного типа
- отсутствующие значения выводного типа заполняются по умолчанию

В стиле Си:

```
int a=5;
float b = (float)a;
int c = 12500;
u_short d = (u_short)c;
```

Static_cast<тип>(цель) возвращает значение выводного типа путем изменения машинного представления значения выводного типа. Выполняется транслятором.

```
short b = static_cast<short>(a);
```

Такой способ нельзя заменить указатель

```
short * c = (static_cast<short*>(&b)
```

Но можно сделать так:

```
void *b = static_cast<void*>(&a);
short* c = static_cast<short*>(b);
```

Подмена имени типа по адресу.

Reinterpret_cast указывает транслятору заменить тип входного значения на указанный, не изменяя адреса.

Результат определен только для конкретной машины

В стиле Си++

static_cast<>();
dynamic_cast<>();
const_cast<>();
reinterpret_cast<>();

- Статическая подмена.

Система типов

Типизация - процесс ограничения числа возможных состояний.

- ограничивает множество типов, над которыми выполняется операция.
- Система типов — статическая
- описывает правила определения данных.

Подстановка имен типов - введение псевдонима для заранее определенного типа. Псевдоним существует в текущей пространстве.

```
typedef unsigned short* PWORD;
typedef int (*func)(int);
typedef uint16_t SHORT;
```

Хранение состояний

Структура - именованный блок данных содержащий переменные любого типа
структура - отдельный тип.

```
struct <имя>
{
  тип имя
  тип имя
  ...
} переменные;
```

- Объявление структуры создает новый тип.
- Поля размещаются в памяти траекторией
- Объявление переменной типа структуры создает изменяемое состояние с отдельным доступом
- Доступ в пространство через разр. имен. → "

Константные данные

Const - константа, которые нельзя изменить

```
const int a = 1; - константа a = 1
int* const p_a = &a; константа указатель на a = 1
```

данные могут изменяться, а адрес указателя нет

Для снятия запрета применяем преобразование.

```
int* p_a = const_cast<int*>(a)
```

Последовательности

#include – команда включения файла
#include <name> – файлы из системных каталогов
#include "name" – файлы из каталогов сборки
#define – определить в файле имя, значение или макрос как угодно
#define NAME
#define const 2
#define MIN(a,b) (((a)<(b))?a:b)

Константы:
__FILE__, __LINE__, __WIN64__, __LINUX__

Макросы:
Команда определяющая вариативный
#define MIN(a,b)...
cout << MIN(4,2) << endl;
Раскрытие: cout << ((4)<(2))?4:2)
Спец символы
преобразует аргумент в строку
конкатенирует две строки
\ перенос на новую строку

Условная компиляция
#ifdef #undef – снятие известности имя #define
#else
#endif #error <сообщение> – выдача ошибки
#elif

Встраиваемые функции.
Начинается словом inline
копирует код функции и прямо подставляет
аргументы, вместо вызова функции
текущую функцию оптимизирует компилятор

Расширение языка:

#pragma once - аналог #ifndef
не рекомендуется
Плюсы и примеры на GitHub