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, сопоставленное адресу, по которому лежат данные с изв. типом. Адрес предоставляет для имени. Операции над ссылками реализует спец. библиотека.
Взятие ссылки - &имя. Предоставляет адрес данных. Обеспечивает многократную подстановку 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 приведет к ошибке
- Ручной контроль использует NULL/nullptr.

```
int a=5; -данные
int *b='&a; - указатель на адрес данных
int &c='a; - ссылка на данные
k b нельзя присвоить значение
k C можно, причем изменится a.
```

Передача аргументов в функцию

- Копирование значения - данные аргумента копир-

нотся внутрь функции.
- По адресу (&) - аргумент это ссылка на данные, откуд во внешней программе
- Копирование адреса (*) + копируется указатель,

Пример передачи на Github.
Доступ к аргументам по адресу
- Доской - если аргумент переменная или ссылка, то получим внешнюю таблицу имен.
- Звездочка - доступ через указатель ≈ ссылке
- Стрелочка + нужен доступ к таблице имен.

Арифметика адресов

- Имя массива - указатель на первый элемент
- Смещение - целое беззнаковое 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 масив
int * array[10];     Масив 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 своя память

Дин. память - вся остальная память вых. машины.
- распределяется между раб. программами на машине
- Требует реализации (на уровне ст. биб 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 терминантом
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; // иниц.
        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 пар, пересечением двух типов
- машинное представление выходного типа
- отсутствующие значения выходного 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>(имя);
```
Возвращает значение выходного типа, полученное путем наложения машинного представления значения входного типа на нулевое значение выходного типа. Выполняется трансляторе.
```
short b = static_cast<short>(a);
```
Таким способом 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.

В стиле Си++
```
static_cast<>();
dynamic_cast<>();
const_cast<>();
reinterpret_cast<>();
```
- Статическая подмена выходного типа, полученная путем наложения машинного представления значения входного типа. Выполняется трансляторе.

reinterpret_cast предписывает транслятору заменить тип входного значения на указанный, не изменяя адреса.

Результат определен только для конкретной машины

Система типов

Типизация - процесс ограничения числа возможных состояний.

- ограничивает множество имен, над которыми выполняется операция.
- система типов – статическая
- описывает правила определения данных.

Подстановка имен типов – введение к сведению 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 друг за другом
- Объявление переменной типа структуры создает изменяемое состояние с отдельным пространством
- Доступ в пространство через разр. имен „→"

Константные данные

const – константы, 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)
```

Препроцессор

#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 известный
#define NAME
#define const 2
#define MIN(a,b) (((a)<(b))?a:b)

Константы:
__FILE__, __LINE__, __WIN64__, __LINUX__

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

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

Встраиваемые функции
Помечается словом inline
копирует блок функции и прямо подставляет аргументы, вместо вызова функции
Такую функцию оптимизирует транслятор.

Расширение языка:

#pragma once - аналог #ifndef
не рекомендуется

Уроки и примеры на GitHub.