ППОИС

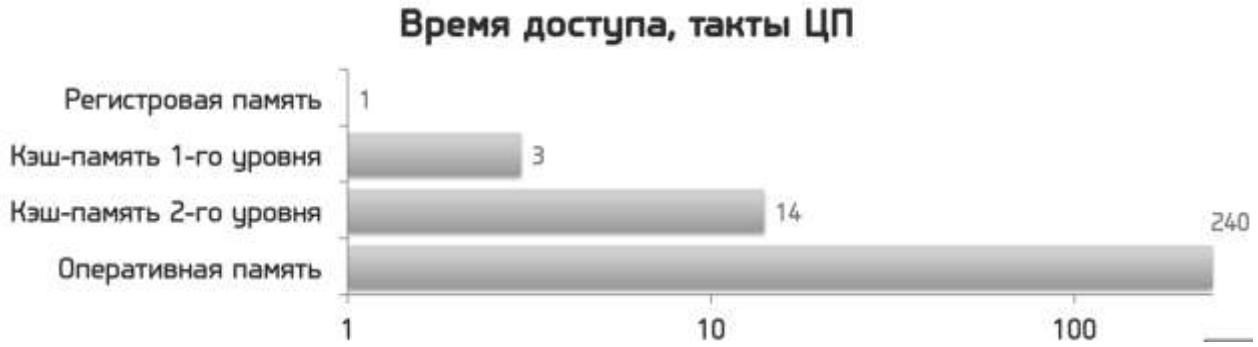
Часть 1

Управление памятью

Содержание

1. Управление памятью на уровне операционной системы
2. Управление памятью на уровне приложения
3. Управление памятью на уровне объектов

Физическая память



Управлению памятью в операционной системе

- Страничная модель
- Виртуальное адресное пространство

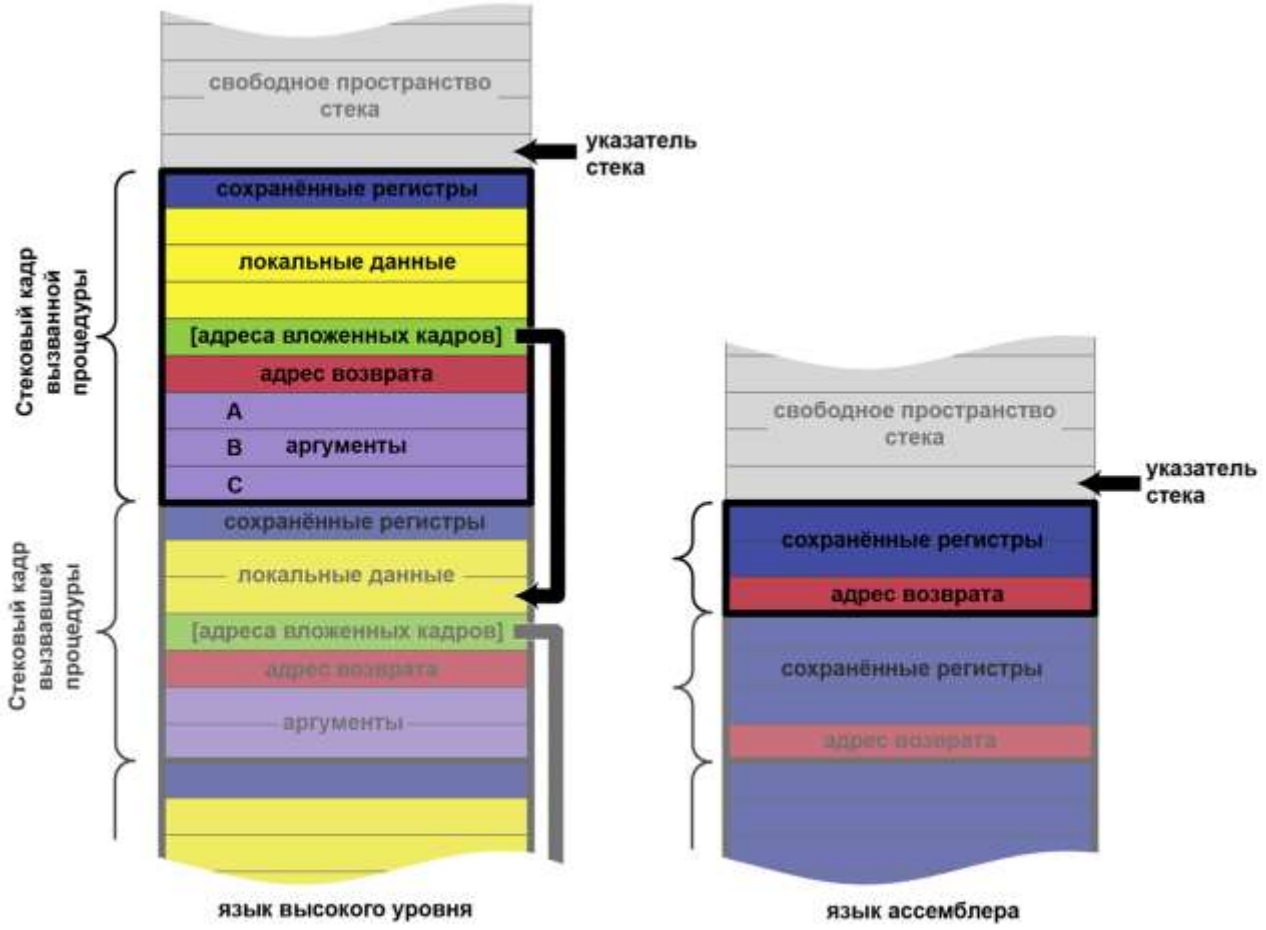
Выделение памяти приложению

- Выделение
- Уничтожение
- Общая память
- Доступ к памяти других процессов

Структура программы

- Сегмент кода
- Сегмент данных
- Сегмент стека

Использование стека: что там хранится и какие ограничения


свободное пространство стека
указатель стека
сохранённые регистры
локальные данные
[адреса вложенных кадров]
адрес возврата
A
B
C
аргументы
Стековый кадр вызванной процедуры
сохранённые регистры
локальные данные
[адреса вложенных кадров]
адрес возврата
аргументы
Стековый кадр вызвавшей процедуры
язык высокого уровня
свободное пространство стека
указатель стека
сохранённые регистры
адрес возврата
сохранённые регистры
адрес возврата
язык ассемблера

Модель кучи (heap)

- менеджеры памяти
- фрагментация
- указатели

Сборщик мусора (Garbage collector)

Процесс автоматического освобождения памяти и удаления объектов, которые уже не используются в программе

- алгоритм выставления флага
- алгоритм подсчета ссылок

Поколения объектов.

Идиома RAII

“Resource Acquisition is Initialization” – захват ресурса есть инициализация

```
class TelephoneCall
{
public:
  TelephoneCall()
  {
     telephoneLine = new TelephoneLine();
     telephoneLine->pickUpThePhoneUp();
  }
  ~TelephoneCall()
  {
     telephoneLine->putThePhoneDown();
     delete telephoneLine;
  }

private:
  TelephoneCall (const TelephoneCall &);
  TelephoneCall& operator=(const TelephoneCall &);
  TelephoneLine* telephoneLine;
};
```

Примеры размещения в памяти

```c
int main() {
    int b = 5;
    return b;
}
```

```c
int a = 5;
int main() {
    return a;
}
```

```c
void func() {
    int c;
    c = 5;
}

int main() {
    func();
    return 0;
}
```

Примеры размещения в памяти

```
int main() {
    int* d = new int;
    *d = 6;
    return *d;
}
```

```
struct Point {
    int x, y;
};

int main() {
    Point p1;
    return 0;
}
```

```
struct Point {
    int x, y;
    Point next;
};

int main() {
    Point* p1 = new Point();
    return 0;
}
```

Примеры размещения в памяти

```
struct Point {
    int x, y;
    Point* next;
};

int main() {
    Point p1;
    p1.next = new Point();
    return 0;
}
```

Выравнивание в памяти

Таблица виртуальных функций

```
1   class shape {
2   public:
3       shape();
4       virtual double area() const;
5       virtual double perimeter() const;
6   private:
7       coordinates position;
8       color outline, fill;
9   };
```

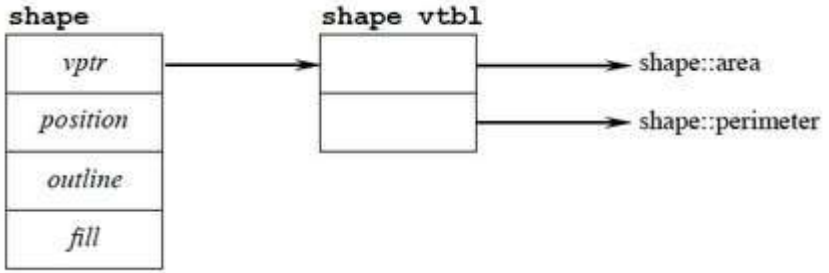
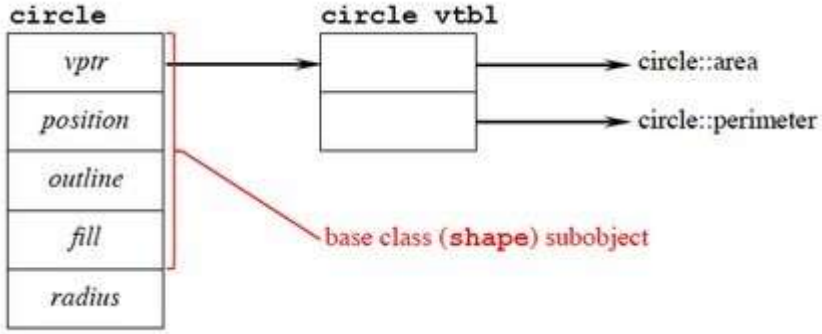


Таблица виртуальных функций

```
11    class circle: public shape {
12    public:
13        circle(double r);
14        virtual double area() const;
15        virtual double perimeter() const;
16    private:
17        double radius;
18    };
```



Инициализация при работе с памятью

Прямую инициализацию:

```
int *ptr1 = new int (7);
```

Uniform-инициализацию (C++11):

```
int *ptr2 = new int { 8 };
```

```
delete ptr;
ptr = nullptr;
```

Временные объекты

```
int sum(int a, int b) {
    return a + b;
}

int main() {
    int c = 5;
    return c + sum(2, 7);
}
```

Копирование объектов

Область видимости

Ссылки

```
int x;

int& y = x;
```

Зачем нужны? Когда не хватает указателей?

“Ссылки в C++ появились чтобы удовлетворить синтаксические потребности механизма перегрузки операторов”

Примеры

```
int n = 0;
int &r = n; /* теперь r -- ссылка на n или второе имя переменной n */
n = 10;
cout << r << '\n'; // выведет 10
r = 20;
cout << n << '\n'; // выведет 20
cout << (&n == &r) << '\n'; // выведет 1, т.е. истина
```

```
int x;
const int &r1 = x; // ссылка на x "только для чтения"
int const &r2 = x; // тоже ссылка на x "только для чтения"
int & const r3 = x; // ошибка компиляции, нельзя ставить const после &
```

Примеры

```
int& max_byref(int& a, int& b)
{
  return a < b? b: a;
}

int main()
{
  int x = 0, y = 0; // собственно имена переменных не обязаны совпадать
  cin >> x >> y;
  max_byref(x, y) = 42;
  cout << "x = " << x << "; y = " << y << '\n';
  return 0;
}
```

Ограничения ссылок

1. Ссылку нельзя переназначить на другой объект.
2. Ссылка должна быть инициализирована при создании
3. Ссылка не может содержать нулевой адрес

Виды ссылок

- Ссылки на неконстантные значения (обычно их называют просто «ссылки» или «неконстантные ссылки»).
- Ссылки на константные значения (обычно их называют «константные ссылки»).
- Ссылки r-value

Понятие l-values и r-values

Свойства выражения

- Модифицируемые l-values (переменная x)
- Немодифицируемые l-values (константа PI)
- Все остальное, r-values:
  - литералы (например, 5)
  - временные значения (например, x + 1)
  - анонимные объекты (например, Fraction(7, 3))

r-values имеют область видимости выражения (уничтожаются в конце выражения, в котором находятся) и им нельзя что-либо присвоить.

Пример

```
class Fraction {
private:
    int m_numerator;
    int m_denominator;

public:
    Fraction(int numerator = 0, int denominator = 1) :
        m_numerator(numerator), m_denominator(denominator) { }

    friend std::ostream& operator<<(std::ostream& out, const Fraction &f1) {
        out << f1.m_numerator << "/" << f1.m_denominator;
        return out;
    }
};

int main() {
    Fraction &&rref = Fraction(4, 7); // ссылка r-value на анонимный объект класса Fraction
    std::cout << rref << '\n';

    return 0;
}
```

Пример

```
class A { };

int main()
{
    const A& a = A();
    A&& b = A();
    return 0;
}
```

Частые области применения r-values

Передача параметров в функцию

Возврат ссылки из функции