Approval system

- Get an approval
- Approve other members
- Presenting your projects

От Си к C++

ВВЕДЕНИЕ В ПРОМЫШЛЕННУЮ СИСТЕМНУЮ РАЗРАБОТКУ

Язык Си

Что свойственно языку Си с точки зрения синтаксиса

- Функции
- Указатели и Массивы
- Ручное управление памятью
- Структуры
- Макросы

Язык Си

Простейшая структура данных на языке Си.

1. Хранение данных в структуре – Небезопасно
2. Функции для работы с СД – Неудобно
3. Функции для копирования СД – Неудобно
4. Вызов функции конструирования и уничтожения памяти для СД – Небезопасно

Проблемы?

Язык Си

Стэк курильщика

```
struct Stack
{
    int canary1;
    float* data;
    int Count;
    int Size;
    int canary2;
};

void    StackConstruct      (struct Stack* StackP);
void    StackDestruct       (struct Stack* StackP);
void    StackPush           (struct Stack* StackP, float num);
float   StackPop            (struct Stack* StackP);
void    StackRegPush        (struct Stack* StackP);
void    StackRegPop         (struct Stack* StackP);
void    StackIncrease       (struct Stack* StackP);
void    DumpStack           (struct Stack StackP);
bool    AssertOk            (struct Stack StackP);
bool    IsStackDestructed   (struct Stack StackP);
```

Что бы нам хотелось?

1) Чтобы компилятор знал какие функции могут работать с объектом стека
2) Чтобы компилятор сам вызывал конструктор и деструктор стека
3) Возможность давать более короткие имена без пересечений
4) Невозможность получить доступ значениям структуры, там, где это не нужно
5) Возможность сделать стек для произвольной структуры данных без void * и уродских макросов
6) Использовать операторы вместо функций

И это только начало!

От Си к C++

- Функции Методы, операторы и перегрузка
- Указатели и Массивы Оболочки и ссылки
- Ручное управление памятью Конструкторы и деструкторы
- Структуры Классы
- Макросы Шаблоны

А также исключения, полиморфизм, наследование, лямбды, RTTI и много-много всего, но об этом позже…

Основы основ C++ и решение проблем языка Си

План рассказа:

- Перегрузка функций
- Перегрузка операторов
- Ссылки
- От структур к классам
- Конструкторы и деструкторы
- Инкапсуляция
- Пространства имен
- Шаблоны
- Наследование и динамический полиморфизм

Перегрузка функций в C++

```
2
3   int foo(int a);
4   int foo(int a, int b);
5
6   int main() {
7       foo(1, 2);
8       foo(1);
9       return 0;
10  }
```

Что будет в языке Си? В C++?

Перегрузка функций в C++ - возможность использовать функций с одним именем и разными входными параметрами

Почему такое невозможно в Си? Ведь компилятор все знает в процессе компиляции

Перегрузка функций в C++

main.cpp

2_lect_c_to_cpp > main.cpp

```cpp
#include <stdio.h>

void functionABC(int a) {
    printf("aa");
}

void functionABC(int a, int b) {
    printf("bb");
}

int main() {
    functionABC(1, 2);
    functionABC(1);
    return 0;
}
```

main.cpp main.c

2_lect_c_to_cpp > main.c

```c
#include <stdio.h>

void functionABC(int a, int b) {
    printf("bb");
}

int main() {
    functionABC(1, 2);
    return 0;
}
```

```
mikita@WSL-laptop:~/course/2_lect_c_to_cpp$ gcc -g main.c
mikita@WSL-laptop:~/course/2_lect_c_to_cpp$ readelf -s a.out | grep "ABC"
    62: 0000000000001149    38 FUNC    GLOBAL DEFAULT   16 functionABC
```

```
mikita@WSL-laptop:~/course/2_lect_c_to_cpp$ g++ -g main.cpp
mikita@WSL-laptop:~/course/2_lect_c_to_cpp$ readelf -s a.out | grep "ABC"
    57: 0000000000001149    35 FUNC    GLOBAL DEFAULT   16 _Z11functionABCi
    60: 000000000000116c    38 FUNC    GLOBAL DEFAULT   16 _Z11functionABCii
```

Перегрузка функций в C++

А что если не все так тривиально?

```
mikita@WSL-laptop:~/course/2_lect_c_to_cpp$ ./a.out
aa
bb
cc
bb
```

```
mikita@WSL-laptop:~/course/2_lect_c_to_cpp$ g++ -g main.cpp
main.cpp: In function ‘int main()’:
main.cpp:21:45: error: call of overloaded ‘functionABC(int, long long unsigned int)’ is ambiguous
   21 |     functionABC(1, (unsigned long long int)2);
      |                                            ^
main.cpp:7:6: note: candidate: ‘void functionABC(int, int)’
    7 | void functionABC(int a, int b) {
      |      ^~~~~~~~~~~
main.cpp:11:6: note: candidate: ‘void functionABC(int, float)’
   11 | void functionABC(int a, float b) {
      |      ^~~~~~~~~~~
```

main.cpp

2_lect_c_to_cpp > main.cpp

```cpp
#include <stdio.h>

void functionABC(int a) {
    printf("aa\n");
}

void functionABC(int a, int b) {
    printf("bb\n");
}

void functionABC(int a, float b) {
    printf("cc\n");
}


int main() {
    functionABC(1);
    functionABC(1, 2);
    functionABC(1, (float)2);
    functionABC(1, (short)2);
    functionABC(1, (unsigned long long int)2);
    return 0;
}
```

Перегрузка функций в C++

Также в перегрузку включены:

- Шаблоны
- Константность
- Пользовательские преобразования типов
- Листы параметров (va_args, …)

Перегрузка не может быть по возвращаемому значению, ключевым словам (inline, noexcept, static), аттрибутам (maybe_unused, nodiscard) и т д

Правила перегрузки могут быть очень сложными, например SFINAE, но это вам рано. Остановимся на интуитивно понятных перегрузках

https://en.cppreference.com/w/cpp/language/overload_resolution

Перегрузка операторов в C++

_lect_c_to_cpp > main.cpp > main()

```cpp
#include <stdio.h>

struct ints {
    int a;
    int b;
};

struct floats {
    float a;
    float b;
};

int main() {
    floats struct_floats{0, 0};
    ints struct_ints{0, 0};

    floats struct_result = struct_floats + struct_ints;
    return 0;
}
```

```cpp
floats operator+(floats struct_floats, ints struct_ints) {
    return {struct_floats.a + struct_ints.a, struct_floats.b + struct_ints.b};
}
```

Перегрузка операторов в C++

```cpp
13 floats operator+(floats struct_floats, ints struct_ints) {
14     return {struct_floats.a + struct_ints.a, struct_floats.b + struct_ints.b};
15 }
16
17 int main() {
18     floats struct_floats{0, 0};
19     ints struct_ints{0, 0};
20
21     floats struct_result = struct_floats + struct_ints;
22     return 0;
23 }
```

```cpp
floats FloatsPlusInts(floats struct_floats, ints struct_ints) {
    return {struct_floats.a + struct_ints.a, struct_floats.b + struct_ints.b};
}

int main() {
    floats struct_floats{0, 0};
    ints struct_ints{0, 0};

    floats struct_result = FloatsPlusInts(struct_floats, struct_ints);
    return 0;
}
```

```
mikita@WSL-laptop:~/course/2_lect_c_to_cpp$ readelf -s a.out | grep "floats"
    54: 0000000000001149    89 FUNC    GLOBAL DEFAULT   16 _Z14FloatsPlusInts6floats
mikita@WSL-laptop:~/course/2_lect_c_to_cpp$ g++ -g main.cpp
mikita@WSL-laptop:~/course/2_lect_c_to_cpp$ readelf -s a.out | grep "floats"
    70: 0000000000001149    89 FUNC    GLOBAL DEFAULT   16 _Zpl6floats4ints
```

Ссылки

Передача аргумента для его модицикации

```
void increase(int i) {
    i++;
}
```

```
int i = 0;

increase(i);
```

```
void increase(int *i) {
    *i++;
}
```

```
int i = 0;

increase(&i);
```

```
void increase(int &i) {
    i++;
}
```

```
int i = 0;

increase(i);
```

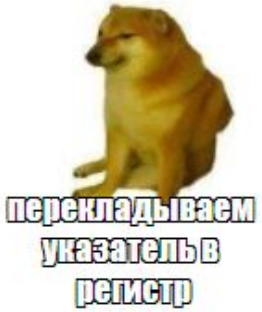
Но какая проблема у ссылок?

Как правило ссылки не используются для передачи параметра, который будет изменен.

Ссылки

Как правило ссылки не используются для передачи параметра, который будет изменен. Но зачем тогда они нужны?

```
struct LotsOfData {
    int arr_int[100500];
    float arr_float[228];
    bool arr_bool[1337];
};
```




```
void ReadLotsOfData(LotsOfData st_data) {
    // do smth with st_data
}
```

```
void ReadLotsOfData(const LotsOfData &st_data) {
    // do smth with st_data
}
```

Ссылки

Передача копий параметров: простейшие типы для чтения

Передача по константной ссылке: любые типы для чтения, больше чем указатель

Передача по указателю: любые данные, для изменения

Можете придумать исключения, когда такое неправильно?

Переход к классам: методы

Вспомним про стек курильщика

```
struct Stack
{
    int canary1;
    float* data;
    int Count;
    int Size;
    int canary2;
};

void    StackConstruct      (struct Stack* StackP);
void    StackDestruct       (struct Stack* StackP);
void    StackPush           (struct Stack* StackP, float num);
float   StackPop            (struct Stack* StackP);
void    StackRegPush        (struct Stack* StackP);
void    StackRegPop         (struct Stack* StackP);
void    StackIncrease       (struct Stack* StackP);
void    DumpStack           (struct Stack StackP);
bool    AssertOk            (struct Stack StackP);
bool    IsStackDestructed   (struct Stack StackP);
```

Что мы тут хотели?

Переход к классам: методы

Простейший пример

```
3   struct Point {
4       int x;
5       int y;
6
7       void Inverse();
8   };
9
10  void Point::Inverse() {
11      this->x *= -1;
12      this->y *= -2;
13  }
14
15  int main() {
16      Point a{2, 3};
17      a.Inverse();
18
19      return 0;
20  }
```

```
1   #include <stdio.h>
2
3   struct Point {
4       int x;
5       int y;
6   };
7
8   void Inverse(Point *this1) {
9       this1->x *= -1;
10      this1->y *= -2;
11  }
12
13  int main() {
14      Point a{2, 3};
15      Inverse(&a);
16
17      return 0;
18  }
```

‘this’ – неявный параметр не статического метода

Переход к классам: конструкторы и

```
3   struct Point {
4       int x_;
5       int y_;
6
7       Point();
8       Point(int x, int y);
9       Point(const Point& another);
10
11      ~Point();
12  };
13
14  Point::Point() : x_(0), y_(0) {}
15
16  Point::Point(int x, int y) {
17      x_ = x;
18      y_ = y;
19  }
20
21  Point::Point(const Point& another) :
22              x_(another.x_), y_(another.y_) {}
23
24  Point::~Point() {
25      // do smth
26  }
```

```
28  int main() {
29      Point a;
30      Point b(1, 2);
31      Point c (a);
32
33      return 0;
34  }
```

Стэк на C++

```
3   struct Stack
4   {
5       Stack();
6       Stack(Stack const & other);
7       ~Stack();
8
9       bool is_empty() const;
10      void push(int data);
11      int pop();
12      int size() const;
13      int & top();
14
15      Stack & operator = (Stack const & other);
16      bool operator == (Stack const & other) const;
17      bool operator != (Stack const & other) const;
18
19      const int START_STACK_SIZE_ = 50;
20
21      int * data_;
22      int size_;
23      int counter_;
24
25  };
```

В дальнейшем:

Лекция, От Си к С++ часть 2

Самостоятельная работа: Изучить правила перегрузки функций, какие бывают перегрузки операторов.
Написать структуру данных на С++, используя то, что мы сегодня рассматривали.

Q&A

Следующая лекция “От Си к C++ часть 2”