Оператор – все, что заканчивается ; Кроме объявления, инициализации

Отладка – прописывание логов

Сложный оператор – несколько простых операторов, объединенных в {}

1. If-else

If(выражение)

{ оператор}

Else

{оператор}

На неравенство нулю можно не проверять

If(x)=If(x!=0)

If (a>0)

If b>0)

Z=a;

Else

Z=b;

Else будет относится к первому оператору

1. Каскад if-else

If(выраж)

{оператор}

Else if (выражение 2)

{оператор}

Else if(выражение3)

{оператор}

Else

{оператор}

1. While(выраж)

{оператор}

Бесконечные циклы

While(1)

{ля-ля-ля}

1. Do while

Do

{O-la-la}

While(выраж)

С нижним окончанием

1. For (выраж1;выраж2;выраж3)

{Шо-то там}

=выраж 1

while(выраж2)

{кукусики

Выраж3}

For(инициализация;условие;инкремент)

**Операторы break и continue**

Continue – переводит на следующую интеракцию цикла

Break выкидывает из всего цикла

1. Switch(выраж) {

Case конст 1; {… break;}

Case конст 2; {… break;}

…

Case конст n; {… break;}

Default {…}

Конст – целые числа

Default – необязательно

**Функции**

Тип имя (параметр1;параметр 2…)//объявление функции

{

Чпок;//определение

Return 1;

}

Void – отсутствие типа, функции ничего не возвращают

Int max(int a; int b)

{

If(a>=b)

Return a;

Return b;

}

Или

Return((a>b)?a:b);

1. Int main()

2. int main(int argc, char argb)

3. int main(…)

Ключевое слово inline – оптимизация функции. Вместо вызова функции, она просто фактически вставляется в код

Inline long long Fact(int n)

{

Long long result=1;

For(int i=1; i<=n;i++)

Result\*=i;

Return result;

}

В цикле for можно объявлять переменные

For(int i=1;…)

**Указатели** – такая переменная, которая принимает значения из адресного пространства процессора, либо нулевое

Адресное пространство зависит от битности системы

Адрес 0x0000

Указатель – тип данных

Int \*

Void \*

Char \*

Double \*

Тип перед звездочкой указывает, какой тип данных хранится по указателю

Указатель, указывающий на адрес 0, не указывает ни на что

Оператор & - возвращает адрес объекта

Int a=2;

Int \*pa=&a;

Int b=\*pa;

Оператор \*- разыменовывание – возвращает значение

Void swap(int \*a; int \*b)

{

Int tmp =a;

A=b;

B=tmp;

}

Если не поставить указатели, swap примет только значения a,b

**Массивы**

Int a[10]

A[n] n<10

Статические массивы хранятся одним куском

A[0]=2;

A[1]=10;

Int \*p=&a[0];

\*p=2

(p+1) – указывает на втторой элемент

\*(p+1)=10

Указатели можно сравнивать

Int strcmp(char \*a, char \*b) // указатель на строку а и b, СДЕЛАТЬ ПРОЩЕ

{

While(\*a!=’\0’&&\*b!=’\0’)

{

If(\*a!=\*b){

Return(\*a<\*b)?-1:1;

}

A++;

B++;

}

If(\*a==\*b)

Return 0;

Else {if(\*a==’\0’){

Return -1;

} else

Return 1;

}

**Структуры и объединения**

Struct SPoint

{

Int x;

Int y;

}

Spoint P;

P.x=1;

P.y=2;

SPoint P= {1,2};

Struct Speoct //прямоугольник

{

SPoint LU;

SPoint PU;

SPoint LD;// точки

}

Объединение

Union U\_tay {

Int value;

Float Tue;

Char chvul;

}

U\_tay t;

**Память**

Память выделяется страницами (~4КБ)

1 часть память зарезервирована под ОС и ядро

Далее сущности:

1. Загруженные библиотеки
2. Программа в виде инструкций
3. Data Segment - переменные
4. Hip(куча)
5. Stack

Стек – структура данных , над которой можно:

1. Проверить, что он пустой
2. Положить на вершину
3. Снять с вершины

*Реализовать стек на C*

Стек нужен для реализации программы и стека вызова функций

int add(int a, int b)

{

Int result = a+b;

Return result;

}

Int main(){

Int answer=add(38,32);

Return 0;

}

Сначала в стек положили мейн(стекфрейм)

Дальше положили переменную answer

Далее – a,b и точка возврата из add

Далее – result из add’a

После – сворачиватся до точки возврата

Далее- до answer’a, после он тоже уйдет и останется только мейн, который уйдет из стека

Stack<<Hip(существенно меньше)

Hip – память занята фрагментировано

Память выделяется дольше, чем в стеке

Работа с Hip:

1. Malloc(объем памяти, который нужно занять из Hip)

calloc

realloc

int i =(int\*)malloc(sizeof(int));

free(i);

i=NULL;

calloc(count, size) – для массива

realloc -если неизвестно, сколько элементов

Компиляция и препроцессоры

Заголовочные .h и исходные .c

Компиляция:

1. Препроцессор
2. Компиляция
3. Компановщик(linker, линковщик)
4. Препроцессоры начинаются с #

#include “ .h” – ищется в директории для поиска заголовочных файлов или в папке исходника

#include < .h> - стандартная библиотека

#include просто вставляет весь код заголовочного в код программы

1. Компиляция

Файлы компилируются независимо друг от друга

1. Компановка

Связь функций из разных файлов

1. Include Guards:

#ifndef NAME\_H

#define NAME\_H

…

#endif

1. #pragma once

#ifdef

#ifndef

#endif

#define

Логирование:

Void Log(\_\_)

{

#ifdef DEBUG\_LOG

Printf();

#else

Fprintf();

#endif

}

Мнимальное число

#define min(a,b) a<b?a:b

Написание макроса(рекомендации):

1. Пишется капсом

**2 семестр**

Книжки: «Язык программирования C++» - Страуструп

Г. Шилт “C++ - полное руководство”

Г. Буч «Объектно-ориентированный анализ и проектирование»

+ С.Майерс, А. Алексондреско, Г. Саттер

Стандарты: 98,2003,11,14,17

**ООП**

Класс – конструкция, хранящая свойства и методы, описывает объект

Class CCar

{

Public:

Void SetAcs(float value);

Void CalcSpeed();

Float Speed();

Float RPM();

Private:

//переменные

}

STL:

Библиотека обобщенных компонент:

1. Контейнеры
2. Обобщенные алгоритмы
3. Итераторы
4. Функциональные объекты
5. Адаптеры
6. Аллокаторы

Гарантии производительности

Контейнеры:

1. vector<T>
2. deque<T>
3. list<T>

Ассоциативные контейнеры:

1. set<key>
2. multiset<key>
3. map<key,T>
4. multimap<key,T>

Обобщенные алгоритмы:

1. Find
2. Max
3. Merge
4. Replace
5. Sort

Итераторы:

1. Указателеобразные объекты
2. Связь между алгоритмами и контейнерами
3. Категории:
4. Выходные
5. Входные
6. Однонаправленные
7. Двунаправленные
8. Произвольного доступа
9. Диапазон итераторов[first,last)
10. Корректный диапазон

Входной итератор:

template<typename InputIterator, typename T>

InputIterator find(InputIterator first, InputIterator last, const T& value)

{

while(first!=last && \*first!=value)

++first;

return first;

}

Входной итератор:

1. Оператор !=
2. Операторы инкремента
3. value=\*iterator
4. operator ==
5. O(1)

Выходной итератор(OutputIteror)

template<typename InputIterator, typename OutputIterator>

OutputIterator copy(InputIterator first, InputIterator last, OutputIterator result)

{

while (first != last )

{

\*result = \*first;

++first;

++result;

}

return result;

}

1. \*Iterator=value
2. Инкремент
3. O(1)

Однонаправленные итераторы(Forward Iterator):

1. Входной
2. Выходной
3. Сохранение для последующего использования

Двунаправленный:

1. Однонаправленный
2. Operator--

C произвольным доступом:

1. Двунаправленный
2. O(1) на любую позицию
3. r,s – итераторы, n-число:
4. r+n, n+r, r-n
5. r[n]=\*(r+n)
6. r+=n, r-=n

Описание контейнеров включает описание итераторов, описание обобщенных алгоритмов включает описание категорий итераторов, с которыми они работают

Итератор вставки(push\_bach, push\_front, insert)

Потоковые операторы(istream\_iterator, ostream\_iterator)

Iterator/const iterator

Обобщенные алгоритмы:

1. неизменяющие
2. изменяющие
3. сортировка
4. обобщенные числовые

# Контейнеры

1. Последовательные
2. vector
3. dequeue
4. list
5. Ассоциативные
6. Map
7. Set

# Адаптеры:

1. Для контейнеров
2. Stack
3. Priority\_Queue
4. Функциональные
5. Bind1st
6. Bind2nd
7. Not1
8. Not2

# Exception

Способы бороться с ошибками:

1. Assert

#include<cassert>

Int main()

{

Assert(2+2==4);

Assert(2+2==5);

Return 0;

}

1. Static\_assert – работает в момент компиляции
2. Исключения

int main

{

Try

{

Foo();

}

Catch(const std::ovetflow\_error& e)

{

//

}

Catch(const std::runtime\_error& e)

{

//

}

Catch(const std::exception& e)

{

//

}

Catch(…)

{

//

}

}

Stack unwinding

1. Сконструированный объект пробрасывается обратно по стеку
2. До встречи подходящего блока try/catch
3. Стек раскручивается обратно
4. Кидать исключения из деструктора не стоит

Инициация исключений – слово throw “class”. Вброшен может быть любой класс

Std::exception

1. Кидать стандартные типы в исключения – неинформативно
2. Исключение несет информацию о событии
3. Std::exception – стандартный класс для исключений

Noexcept

1. Гарантирует, что функция не кидает исключения
2. Не сворачивает стек
3. Позволяет компилятору лучше оптимизировать код
4. Std::terminate

C++ 11

1. Auto
2. Range for

Std::vector<int> v(10)

Std::generate(v.begin(),v.end(),random);

For(auto x:v)

Std::cout<<x%100<<’ ‘;

Std::cout<<std::endl;

For(auto&x:v)

X=1;

1. Initialize list
2. Uniform initialization
3. Variadic Templates

…